

版权注意事项：

- 1、书籍版权归作者和出版社所有
- 2、本PDF仅限用于个人获取知识，进行私底下的知识交流
- 3、PDF获得者不得在互联网上以任何目的进行传播
- 4、如觉得书籍内容很赞，请购买正版实体书，支持作者
- 5、请于下载PDF后24小时内删除本PDF。

大数据科学与应用丛书

- 深入分析组件原理 ·
- 充分展示搭建过程 ·
- 详细指导应用开发 ·

Hadoop

大数据实战权威指南

黄东军 编著



中国工信出版集团



电子工业出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
<http://www.phei.com.cn>

大数据科学与应用丛书

Hadoop

大数据实战权威指南

黄东军 编著

电子工业出版社

Publishing House of Electronics Industry

北京·BEIJING

内 容 简 介

大数据贵在落实!

本书是一本讲解大数据实战的图书,按照“深入分析组件原理、充分展示搭建过程、详细指导应用开发”编写。全书分为三篇,第一篇为大数据的基本概念和技术,主要介绍大数据的背景、发展及关键技术;第二篇为 Hadoop 大数据平台搭建与基本应用,内容涉及 Linux、HDFS、MapReduce、YARN、Hive、HBase、Sqoop、Kafka、Spark 等;第三篇为大数据处理与项目开发,包括交互式数据处理、协同过滤推荐系统、销售数据分析系统,并就京东的部分销售数据应用大数据进行处理分析。

本书适合初学者入门和进阶,也可供希望全面、系统地理解并掌握大数据实际应用的读者参考,对从事大数据项目开发的专业人员也有参考价值。

为了方便读者实践,本书配有开发资源包,读者可登录华信教育资源网(www.hxedu.com.cn)免费注册后下载。

未经许可,不得以任何方式复制或抄袭本书之部分或全部内容。

版权所有,侵权必究。

图书在版编目(CIP)数据

Hadoop 大数据实战权威指南 / 黄东军编著. — 北京: 电子工业出版社, 2017.7
(大数据科学与应用丛书)

ISBN 978-7-121-31821-4

I. ①H… II. ①黄… III. ①数据处理软件—指南 IV. ①TP274-62

中国版本图书馆 CIP 数据核字(2017)第 129534 号

责任编辑: 田宏峰

印 刷: 三河市鑫金马印装有限公司

装 订: 三河市鑫金马印装有限公司

出版发行: 电子工业出版社

北京市海淀区万寿路 173 信箱 邮编: 100036

开 本: 787×980 1/16 印张: 23.5 字数: 526 千字

版 次: 2017 年 7 月第 1 版

印 次: 2017 年 7 月第 1 次印刷

定 价: 68.00 元

凡所购买电子工业出版社图书有缺损问题, 请向购买书店调换。若书店售缺, 请与本社发行部联系, 联系及邮购电话: (010) 88254888, 88258888。

质量投诉请发邮件至 zltz@phei.com.cn, 盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式: tianhf@phei.com.cn。

前 言

本书内容

本书分为三篇，共有 12 章。

第一篇 大数据的基本概念和技术

第 1 章 绪论，描述大数据的时代背景与国家大数据战略，探讨大数据的概念和特性，重点阐述大数据支撑体系，包括数据采集、存储、分布式计算和应用，并讨论大数据人才特点与能力要求。

第 2 章 Hadoop 大数据关键技术，详细介绍大数据系统涉及的主流技术，主要包括数据采集与生成、数据分布式存储、分布式计算框架、数据分析与挖掘等方面的技术和工具。

第二篇 Hadoop 大数据平台搭建与基本应用

第 3 章 Linux 操作系统与集群搭建，介绍 Linux 集群的安装、Java 开发包 JDK 的安装，以及集群的配置方法。

第 4 章 HDFS 安装与基本应用，介绍 Hadoop HDFS 的架构、工作原理，以及 Hadoop 安装、配置、启动和程序的运行。

第 5 章 MapReduce 与 YARN，介绍 MapReduce 的工作原理，描述 MapReduceV2（也就是 YARN）的架构和执行流程。本章重点介绍如何设计 MapReduce 程序，给出了在 Eclipse 中实现 Java 语言 MapReduce 程序的具体过程。

第 6 章 Hive 和 HBase 的安装与应用，主要介绍 Hive 和 HBase 的安装配置和应用方法，同时也介绍 MySQL 和 ZooKeeper 的安装与应用。

第 7 章 Sqoop 和 Kafka，介绍 Sqoop 和 Kafka 组件的安装及其基本应用方法。

第 8 章 Spark 集群安装与开发环境配置，介绍 Spark 架构及其工作原理，详细介绍 Spark 开发环境的安装与配置，包括热门的 IntelliJ IDEA 集成开发环境的安装与基本应用。

第 9 章 Spark 应用基础, 介绍 Spark 程序的运行模式和应用设计方法, 通过编写计算圆周率 π 、基于随机森林模型的贷款风险预测 Scala 程序, 展示了在集成开发环境 IDEA 中编写 Spark 程序的流程。

第三篇 大数据处理与项目开发

第 10 章 交互式数据处理, 介绍如何利用 Hive 进行大数据处理和分析。Hive 是建立在 Hadoop MapReduce 基础上的数据仓库工具, 用户借助 SQL 语句, 可完成很多处理和分析, 因此, 对实际工作者有很大帮助。

第 11 章 协同过滤推荐系统, 介绍推荐算法的基本概念和应用, 展示基于 Spark 的机器学习库 MLlib 实现的协同推荐应用。

第 12 章 销售数据分析系统, 通过一个完整的销售数据分析系统设计, 展示如何利用 Hadoop 的各种组件开发实际的大数据应用系统。本章运用到的组件包括 HDFS、MySQL、Eclipse、Phoenix、HBase、WebCollector、Servlet、Tomcat 等, 所展示的数据和应用均来自真实场景, 对读者有较高参考价值。

本书特点

把原理、架构、运行流程分析与实际应用融合起来介绍, 融合性阐述框架优于单纯的原理分析, 因为原理最终要付诸应用。

本书高度重视实践能力的培养, 对系统安装、配置和应用过程给出了十分详细的描述, 所有实验都是基于实际完成的操作介绍的, 并配有现场截图, 为读者展示了真实、详尽、可重现的场景, 十分方便读者自学和钻研。

与很多大数据技术书籍不同, 本书突出了数据处理本身, 深入介绍了如何运用技术进行实际的数据分析, 所采用的数据样本来自生产一线, 所展示的项目具有实用的参考价值, 读者掌握这些技术之后, 就可以开始进行项目开发了。

本书的读者群

本书十分适合初学者入门和进阶。

本书也可供那些已经学习过 Hadoop 组件技术, 但希望全面、系统地理解并掌握实际应用的读者参考。

本书对从事大数据项目开发的专业人员也有参考价值, 书中所描述的 Hadoop 组件应用中遇到的各种问题及其解决办法, 十分实用。

本书特别适合自学，读者完全可以利用本书给出的资源和示例，一步一步地完成各项操作和应用，体验一种登堂入室的成就感。

致谢

感谢大数据时代，感谢开源社区，感谢 Apache 基金会，感谢 Google，感谢所有关心和热爱大数据的人们！

作者在创作本书中借鉴了中科普开（北京）科技公司的部分培训资源，在此谨表示衷心的感谢。特别感谢中南大学郑瑾副教授，本书的部分内容使用了她编撰的书稿。由衷地感谢王建新教授、李建彬教授、张祖平教授，他们耐心地审阅了本书，提出了中肯的意见和建议。非常感谢电子工业出版社田宏峰编辑，他细心专业的工作方式，给作者留下深刻印象，并为本书的高质量印装提供了保障。

由于作者水平有限，本书的错误和疏漏在所难免，恳请广大读者提出宝贵意见和建议。联系邮箱：djhuang@csu.edu.cn。

作 者

2017 年 6 月于长沙

目 录

第一篇 大数据的基本概念和技术

第 1 章 绪论	3
1.1 时代背景	3
1.1.1 全球大数据浪潮	3
1.1.2 我国的大数据国家战略	5
1.2 大数据的概念	7
1.2.1 概念	7
1.2.2 特征	8
1.3 技术支撑体系	9
1.3.1 概览	9
1.3.2 大数据采集层	9
1.3.3 大数据存储层	10
1.3.4 大数据分析（处理与服务）层	11
1.3.5 大数据应用层	11
1.3.6 垂直视图	13
1.4 大数据人才及其能力要求	14
1.4.1 首席数据官	14
1.4.2 数据科学家（数据分析师）	15
1.4.3 大数据开发工程师	16
1.4.4 大数据运维工程师	17
1.5 本章小结	17
第 2 章 Hadoop 大数据关键技术	19
2.1 Hadoop 生态系统	19
2.1.1 架构的基本理论	19
2.1.2 主要组件及其关系	21

2.2	数据采集	24
2.2.1	结构化数据采集工具	24
2.2.2	日志文件采集工具与技术	25
2.3	大数据存储技术	29
2.3.1	相关概念	29
2.3.2	分布式文件存储系统	34
2.3.3	数据库与数据仓库	38
2.4	分布式计算框架	43
2.4.1	离线计算框架	43
2.4.2	实时流计算平台	50
2.5	数据分析平台与工具	57
2.5.1	面向大数据的数据挖掘与分析工具	57
2.5.2	机器学习	61
2.6	本章小结	66
<h2>第二篇 Hadoop 大数据平台搭建与基本应用</h2>		
第 3 章	Linux 操作系统与集群搭建	69
3.1	Linux 操作系统	69
3.1.1	概述	69
3.1.2	特点	70
3.1.3	Linux 的组成	72
3.2	Linux 安装与集群搭建	75
3.2.1	安装 VMware Workstation	75
3.2.2	在 VMware 上安装 Linux (CentOS7)	79
3.3	集群的配置	91
3.3.1	设置主机名	91
3.3.2	网络设置	93
3.3.3	关闭防火墙	98
3.3.4	安装 JDK	99
3.3.5	免密钥登录配置	102
3.4	Linux 基本命令	105
3.5	本章小结	112
第 4 章	HDFS 安装与基本应用	113
4.1	HDFS 概述	113

4.1.1	特点	113
4.1.2	主要组件与架构	114
4.2	HDFS 架构分析	114
4.2.1	数据块	114
4.2.2	NameNode	115
4.2.3	DataNode	116
4.2.4	SecondaryNameNode	117
4.2.5	数据备份	117
4.2.6	通信协议	118
4.2.7	可靠性保证	118
4.3	文件操作过程分析	119
4.3.1	读文件	119
4.3.2	写文件	120
4.3.3	删除文件	122
4.4	Hadoop HDFS 安装与配置	122
4.4.1	解压 Hadoop 安装包	122
4.4.2	配置 Hadoop 环境变量	123
4.4.3	配置 Yarn 环境变量	124
4.4.4	配置核心组件文件	125
4.4.5	配置文件系统	125
4.4.6	配置 yarn-site.xml 文件	126
4.4.7	配置 MapReduce 计算框架文件	128
4.4.8	配置 Master 的 slaves 文件	129
4.4.9	复制 Master 上的 Hadoop 到 Slave 节点	129
4.5	Hadoop 集群的启动	130
4.5.1	配置操作系统环境变量	130
4.5.2	创建 Hadoop 数据目录	131
4.5.3	格式化文件系统	132
4.5.4	启动和关闭 Hadoop	133
4.5.5	验证 Hadoop 是否启动成功	133
4.6	Hadoop 集群的基本应用	136
4.6.1	HDFS 基本命令	136
4.6.2	在 Hadoop 集群中运行程序	139
4.7	本章小结	141

第 5 章	MapReduce 与 Yarn	143
5.1	MapReduce 程序的概念	143
5.1.1	基本编程模型	143
5.1.2	计算过程分析	144
5.2	深入理解 Yarn	147
5.2.1	Yarn 的基本架构	147
5.2.2	Yarn 的工作流程	151
5.3	在 Linux 平台安装 Eclipse	152
5.3.1	Eclipse 简介	153
5.3.2	安装并启动 Eclipse	154
5.4	开发 MapReduce 程序的基本方法	155
5.4.1	为 Eclipse 安装 Hadoop 插件	156
5.4.2	WordCount: 第一个 MapReduce 程序	160
5.5	本章小结	175
第 6 章	Hive 和 HBase 的安装与应用	177
6.1	在 CentOS7 下安装 MySQL	177
6.1.1	下载或复制 MySQL 安装包	177
6.1.2	执行安装命令	178
6.1.3	启动 MySQL	179
6.1.4	登录 MySQL	179
6.1.5	使用 MySQL	181
6.1.6	问题与解决办法	182
6.2	Hive 安装与应用	183
6.2.1	下载并解压 Hive 安装包	183
6.2.2	配置 Hive	184
6.2.3	启动并验证 Hive	187
6.2.4	Hive 的基本应用	189
6.3	ZooKeeper 集群安装	190
6.3.1	ZooKeeper 简介	190
6.3.2	安装 ZooKeeper	191
6.3.3	配置 ZooKeeper	191
6.3.4	启动和测试	193
6.4	HBase 的安装与应用	195

6.4.1	解压并安装 HBase	195
6.4.2	配置 HBase	196
6.4.3	启动并验证 HBase	199
6.4.4	HBase 的基本应用	200
6.4.5	应用 HBase 中常见问题及其解决办法	203
6.5	本章小结	204
第 7 章	Sqoop 和 Kafka 的安装与应用	205
7.1	安装部署 Sqoop	205
7.1.1	下载或复制 Sqoop 安装包	205
7.1.2	解压并安装 Sqoop	206
7.1.3	配置 Sqoop	206
7.1.4	启动并验证 Sqoop	208
7.1.5	测试 Sqoop 与 MySQL 的连接	209
7.2	安装部署 Kafka 集群	211
7.2.1	下载或复制 Kafka 安装包	211
7.2.2	解压缩 Kafka 安装包	211
7.2.3	配置 Kafka 集群	211
7.2.4	Kafka 的初步应用	213
7.3	本章小结	218
第 8 章	Spark 集群安装与开发环境配置	219
8.1	深入理解 Spark	219
8.1.1	Spark 系统架构	219
8.1.2	关键概念	221
8.2	安装与配置 Scala	224
8.2.1	下载 Scala 安装包	225
8.2.2	安装 Scala	225
8.2.3	启动并应用 Scala	226
8.3	Spark 集群的安装与配置	226
8.3.1	安装模式	226
8.3.2	Spark 的安装	227
8.3.3	启动并验证 Spark	230
8.3.4	几点说明	234
8.4	开发环境安装与配置	236

8.4.1	IDEA 简介	236
8.4.2	IDEA 的安装	236
8.4.3	IDEA 的配置	238
8.5	本章小结	243
第 9 章	Spark 应用基础	245
9.1	Spark 程序的运行模式	245
9.1.1	Spark on Yarn-cluster	245
9.1.2	Spark on Yarn-client	246
9.2	Spark 应用设计	247
9.2.1	分布式估算圆周率	248
9.2.2	基于 Spark MLlib 的贷款风险预测	265
9.3	本章小结	285

第三篇 数据处理与项目开发术

第 10 章	交互式数据处理	289
10.1	数据预处理	289
10.1.1	查看数据	289
10.1.2	数据扩展	291
10.1.3	数据过滤	292
10.1.4	数据上传	293
10.2	创建数据仓库	294
10.2.1	创建 Hive 数据仓库的基本命令	294
10.2.2	创建 Hive 分区表	296
10.3	数据分析	299
10.3.1	基本统计	299
10.3.2	用户行为分析	301
10.3.3	实时数据	303
10.4	本章小结	304
第 11 章	协同过滤推荐系统	305
11.1	推荐算法概述	305
11.1.1	基于人口统计学的推荐	305
11.1.2	基于内容的推荐	306
11.1.3	协同过滤推荐	307

11.2	协同过滤推荐算法分析	308
11.2.1	基于用户的协同过滤推荐	308
11.2.2	基于物品的协同过滤推荐	310
11.3	Spark MLlib 推荐算法应用	312
11.3.1	ALS 算法原理	312
11.3.2	ALS 的应用设计	315
11.4	本章小结	329
第 12 章	销售数据分析系统	331
12.1	数据采集	331
12.1.1	在 Windows 下安装 JDK	331
12.1.2	在 Windows 下安装 Eclipse	334
12.1.3	将 WebCollector 项目导入 Eclipse	335
12.1.4	在 Windows 下安装 MySQL	336
12.1.5	连接 JDBC	339
12.1.6	运行爬虫程序	340
12.2	在 HBase 集群上准备数据	342
12.2.1	将数据导入到 MySQL	342
12.2.2	将 MySQL 表中的数据导入到 HBase 表中	344
12.3	安装 Phoenix 中间件	347
12.3.1	Phoenix 架构	347
12.3.2	解压安装 Phoenix	348
12.3.3	Phoenix 环境配置	349
12.3.4	使用 Phoenix	350
12.4	基于 Web 的前端开发	353
12.4.1	将 Web 前端项目导入 Eclipse	353
12.4.2	安装 Tomcat	355
12.4.3	在 Eclipse 中配置 Tomcat	355
12.4.4	在 Web 浏览器中查看执行结果	359
12.5	本章小结	361

第一篇

大数据的基本概念和技术

最早提出“大数据”时代到来的全球知名咨询公司麦肯锡称：“数据，已经渗透到当今每一个行业和业务职能领域，成为重要的生产因素。人们对于海量数据的挖掘和运用，预示着新一波生产率增长和消费者盈余浪潮的到来”。

本章主要分析大数据的时代背景与国家大数据战略，给出大数据的概念，并分析其特性，重点介绍大数据技术支撑体系，包括数据采集、存储、分布式计算和应用，最后简要讨论大数据人才特点与能力要求。

1.1 时代背景

1.1.1 全球大数据浪潮

为什么最近几年里大数据变得如此引人注目？大数据到底有多大？

一组名为“互联网上一天”的数据告诉我们，一天之中，互联网产生的全部内容可以刻满 1.68 亿张 DVD；发出的邮件有 2940 亿封之多；发出的社区帖子达 200 万个（相当于《时代》杂志 770 年的文字量）；卖出的手机为 37.8 万台，高于全球每天出生的婴儿数量 37.1 万。

目前，全球数据量已经从 TB（ $1024\text{ GB}=1\text{ TB}$ ）级别跃升到 PB（ $1024\text{ TB}=1\text{ PB}$ ）、EB（ $1024\text{ PB}=1\text{ EB}$ ）乃至 ZB（ $1024\text{ EB}=1\text{ ZB}$ ）级别。国际数据公司（IDC）的研究结果表明，2008 年全球产生的数据量为 0.49 ZB，2009 年的数据量为 0.8 ZB，2010 年增长到 1.2 ZB，2011 年的数量更是高达 1.82 ZB，相当于全球每人产生 200 GB 以上的数据。而

到 2016 年,人类生产的所有印刷材料的数据量是 300 PB,全人类历史上说过的所有话的数据量大约是 5 EB。IBM 的研究称,整个人类文明所获得的全部数据中,有 90%是过去两年内产生的。而到了 2020 年,全世界所产生的数据规模将达到今天的 44 倍。

这样的趋势将会持续下去。我们现在还处于大数据的初级阶段,随着技术的进步,我们的设备、交通工具和迅速发展的“可穿戴”科技将能互连互通。科技的进步已经使创造、采集和管理信息的成本降至十年前的六分之一,而从 2005 年起,用在硬件、软件、人才及服务之上的商业投资也增长了整整 50%,达到了 4000 亿美元。

正如《纽约时报》2012 年 2 月的一篇专栏文章所称,“大数据”时代已经降临,在商业、经济及其他领域中,决策将日益基于数据和分析而做出,而并非基于经验和直觉。哈佛大学社会学教授加里金说:“这是一场革命,庞大的数据资源使得各个领域开始了量化进程,无论学术界、商界还是政府,所有领域都将开始这种进程。”

越来越多的政府、企业等机构开始意识到数据正在成为组织最重要的资产,数据分析能力正在成为组织的核心竞争力。

2012 年 3 月 22 日,美国政府宣布投资 2 亿美元拉动大数据相关产业发展,将“大数据战略”上升为国家意志。美国政府将数据定义为“未来的新石油”,并表示一个国家拥有数据的规模、活性及解释运用的能力将成为综合国力的重要组成部分,未来,对数据的占有和控制甚至将成为陆权、海权、空权之外的另一种国家核心资产。

联合国也在 2012 年发布了大数据政务白皮书,指出大数据对于联合国和各国政府来说是一个历史性的机遇,人们如今可以使用极为丰富的数据资源,来对社会经济进行前所未有的实时分析,帮助政府更好地响应社会和经济运行。

最为积极的还是众多的 IT 企业。麦肯锡在一份名为“大数据,是下一轮创新、竞争和生产力的前沿”的专题研究报告中提出,“对于企业来说,海量数据的运用将成为未来竞争和增长的基础”,该报告在业界引起广泛反响。麦肯锡的报告发布后,大数据迅速成为了计算机行业争相传诵的热门概念,也引起了包括金融界在内的各行各业的高度关注。随着互联网技术的不断发展,数据本身是资产,这一点在业界已经形成共识。如果说云计算为数据资产提供了保管、访问的场所和渠道,那么如何盘活数据资产,使其为国家治理、企业决策乃至个人生活服务,则是大数据的核心议题,也是云计算内在的灵魂和必然的升级方向。事实上,全球互联网巨头都已意识到了“大数据”时代,数据的重要意义,包括谷歌、苹果、惠普、IBM、微软在内的全球 IT 巨头纷纷通过收购“大数据”相关厂商来实现技术整合,可见其对“大数据”的重视。

例如, IBM 提出, 上一个十年, 他们抛弃了 PC, 成功转向了软件和服务, 而这次将远离服务与咨询, 更多地专注于因大数据分析软件而带来的全新业务增长点。IBM 执行总裁罗睿兰认为, “数据将成为一切行业当中决定胜负的根本因素, 最终数据将成为人类至关重要的自然资源。”

在国内, 阿里巴巴在大数据应用和开发上投入巨资, 已经取得了令人瞩目的成绩; 百度也致力于开发自己的大数据处理和存储系统; 腾讯则提出要开创数据化运营的黄金时期, 把整合数据看成未来的关键任务。

总体上, 自从 SGI 的首席科学家 John R. Masey 在 1998 年提出大数据概念, 到大数据分析技术广泛应用于社会的各个领域, 已经走过了近 20 年的时间。现在, 再也没有人怀疑大数据分析的力量, 并且都在竞相利用大数据来增强自己企业的业务竞争力。但是, 即使近 20 年已经过去, 大数据分析行业仍然处于快速发展的初期, 每时每刻都在产生新的变化, 特别是随着移动互联网的快速发展, 大数据从概念到实用、从结构化数据分析到非结构化数据分析, 正处于新的高潮和进化阶段。

1.1.2 我国的大数据国家战略

毫无疑问, 在全世界进入以信息产业为主导的新经济发展时期, 以大数据为代表的新兴产业将以新经济方式引领新常态, 我们国家也必然提出并实施国家大数据发展战略。

党的十八届五中全会提出了“十三五”规划纲要对实施网络强国战略、互联网+行动计划和大数据战略等做了部署。纲要提出要切实贯彻落实好着力推动互联网和实体经济深度融合, 以信息流带动技术流、资金流、人才流、物资流, 促进资源的配置优化, 促进全要素生产力的提升, 为推动创新发展、转变经济增长方式、调整经济结构发挥重要作用。

为了贯彻落实十八届五中全会精神, 国务院于 2015 年 8 月出台了促进大数据发展行动纲要, 提出要通过开放、产业和安全“三位一体”建设数据强国。三位一体主要是政府数据开放共享, 它是开放的条件; 产业是根基, 即以推动产业创新发展为根本; 安全是保障, 要健全数据的安全保障体系。

这个行动纲要总的来讲可以概括为“一个目标, 三大内容、十项工程、七大举措”。一个目标, 就是全面推进我国大数据发展和应用, 加快建设数据强国目标。三大内容, 主要是包括加快政府数据的开放, 推动资源整合提升的治理能力; 推动产业创新发展, 培育新业态, 助力经济转型; 健全安全保障体系, 强化安全支撑, 提高管理的水平。十项工程, 主要包括政府数据、国家大数据的资源统筹发展工程, 政府治理大数据工程, 公共服务大数据工程, 工业和新兴产业大数据工程, 现代农业大数据工程, 万众

创新大数据工程，等等。七个举措，包括完善组织机制，加快法规制度建设，健全市场发展机制，建立标准规范体系，加大财政金融的支持，加强专业人士的培养，促进国际交流合作。

同时，国务院还决定建立国家大数据发展和应用的统筹协调机制，通过设立 3+X 工作机制，主要是由工业和信息化部、国家发展和改革委员会，以及中央网络安全和信息化领导小组办公室（网信办）三个部门牵头，联合其他 40 个政府部门建立的促进大数据发展的部级联席会议制度。围绕着三个关键的环节，精准发力，主要包括加快数据的开放共享，推动产业的创新发展，科学规范应用的数据。

当前，在大数据产业的发展思路方面，国家强调要以市场为导向，并在四个方面进行探索：一是支持关键技术产品的研发和产业化，二是推动行业大数据应用的不断深化，三是繁荣大数据的产业生态，四是完善大数据的支撑体系。在关键技术产品的研发和产业化方面，主要就是要抓住大数据引领的 IT 技术的架构和产业变革的机遇，在技术研发、产品体系、服务支撑这三个方面着力。在推动行业大数据应用不断深化方面，主要是大力推动工业和信息通信业大数据应用发展，支持大数据跨行业的融合应用发展，包括公共服务、社会治理、金融、能源、交通和农业等国民经济各个方面。在繁荣大数据产业生态方面，主要是合理规划大数据的基础设施建设，促进大数据的创业创新发展，优化大数据产业的区域布局。

大数据贵在落实。国家有关部门近期关于大数据方面的重点工作包括：

第一就是支持大数据关键产品的研发和产业化，目前在三个方面，进一步加大支持力度，首先是利用项目资金支持研发和产业化；其次在技术产品研发领域，与产业化之间非结构化的数据处理，大数据管理系统及数据分析，数据安全等关键技术产品的研发和产业化；然后是形成一批自主创新、技术先进，满足重大应用需求的产品解决方案。

第二是大力推动工业大数据的应用，在落实中国制造 2025、互联网+等战略中，将大数据作为抓手，促进大数据与工业融合创新发展。启动智能制造试点示范的专项行动，利用相关的项目资金支持工业大数据技术产品的应用示范。加强制造企业与信息服务企业的合作，推进大数据在研发、设计、生产、制造、售后服务等全生命周期的应用，形成工业大数据优秀的解决方案，推动工业大数据与工业云、工业互联网、CPS 等协调发展。

第三是支持地方开展大数据产业发展的应用试点，包括鼓励和支持各地方、各行业、各部门先行先试，开展大数据方面的探索和实践，严格标准，固定质量，按照成熟一个建

设一个的原则,选择有条件、有基础的地方和区域建设大数据综合实验区,鼓励并支持地方结合自身的基础和优势,发展大数据产业,推动大数据产业的集聚发展。

第四是推动大数据的标准体系建设,依托大数据标准工作组,加快数据质量、数据安全、数据开放共享和交易等标准的研制工作,结合大数据综合实验区和产业集聚区,选择一批关键急需的标准在综合实验区开展应用试点,积极参与国际标准的制定工作,推进标准的国际化,提高我国大数据标准制定的国际话语权。

1.2 大数据的概念

大数据是当前的热门话题,人人都在谈论它,但我们需要从纷繁的议论中看到事物的本质。究竟什么是大数据?它的本质特征是什么?大数据在技术上包含哪些内容?怎样让大数据应用落地?本节介绍大数据的基本概念,探讨大数据的基本特征,阐述大数据的技术支撑体系。

1.2.1 概念

在舍恩伯格及库克耶编写的《大数据时代》一书中,大数据被定义为不用随机分析法(抽样调查)这样的捷径,而采用全量模式进行分析处理的数据。

事实上,有关大数据的定义目前并没有一个统一的说法,这也反映出大数据作为快速发展中事物的特点。以下是几个比较典型的大数据定义。

维基百科:大数据是指无法在一定时间内用常规软件工具对其内容进行采集、存储、处理和应用的数据集合。

百度:大数据(**big data, mega data**),或称巨量资料,指的是需要新处理模式才能产生更强大决策力、洞察力和流程优化能力的海量、高增长率和多样化的信息资产。而大数据技术,则是指从各种各样类型的大数据中,快速获得有价值信息的方法或能力。

我们认为,大数据是互联网发展到一定阶段后,数据爆炸性增长的一种态势,这种态势具有强烈的时代特征。所以,给大数据下定义,不能脱离互联网,也需要包含以云计算为代表的技术创新。因此,我们给出的大数据概念是:大数据是指在互联网和以大规模分布式计算为代表的平台支持下被采集、存储、分析和应用的具有产生更高决策价值的巨量、高增长率和多样化的信息资产。显然,这个定义更加全面和准确。

1.2.2 特征

为了深入理解大数据的概念，有必要分析一下大数据的基本特征。

目前，人们普遍采用 4V 表示大数据的特征：Volume（大量）、Velocity（高速）、Variety（多样）、Value（价值）。下文解释了 4V 的含义。

大量，就是指数量巨大。第 1 章已经给出了很多实例。确实，互联网上的数据每年增长 50%，每两年便将翻一番，而目前世界上 90% 以上的数据是最近几年才产生的。据 IDC 预测，到 2020 年全球将总共拥有 35 ZB 的数据量。互联网是大数据发展的前提，随着 Web 2.0 时代的发展，人们似乎已经习惯了将自己的生活通过网络进行数据化，方便分享以及记录并回忆。这里有必要指出，巨量是大数据的首要特性。在很多场合，少量数据就有很高的应用价值，但是，这并不表示数据越少越好，少量有价值的信息是从大数据中挖掘出来的，没有大数据，就没有这些小数据。大数据时代，决策被置于全量式和全景式的环境下。

高速，这是大数据的关键特性。高速性本质是在线（online），这并不意味着绝对速度快，真正有革命意义的是数据变得在线了，这恰恰是互联网的特点。所有数据在线这个事情，远比“大”更反映本质。例如“优步”系统需要大量交通数据支持，如果这些数据是离线的，就没有什么用。为什么今天淘宝数据值钱，因为在线，写在纸上或磁带上的数据效率极其低下。其实，大数据以前也有，但仅仅只有数据大是没有用处的，例如，欧洲粒子物理对撞实验室做一次碰撞产生的数据是巨大的，如果不采用在线分布式并行处理，恐怕无法获得有意义的实验结果。

多样，表示来源与形态具有包罗万象的特点，这是大数据的自然属性，因为人类生活本身是极具多样性的。目前，由网络日志、条码与射频识别（RFID）、传感器网络、工业生产过程、政府社会管理、社交网络、互联网文本和文件、互联网搜索引擎、呼叫详细记录、视频监控、天气预报、基因测序、军事侦察、医疗记录、音影档案、银行交易记录、大规模电子商务等系统或活动产生的数据，已经成为大数据的主要来源。

价值，一方面指数据即生产力，即具有决策价值，被比喻为新时代的石油和黄金；另一方面，也表示大数据的价值密度很低。例如，几小时的监控视频中可能有价值的就两秒钟，其价值需要通过数据过滤、清洗、挖掘和呈现等多个处理步骤才能展现出来。必须理解，价值是大数据的基本属性，人类决策历来依靠数据，但是在大数据时代，数据的决策价值得到了空前的提升。

1.3 技术支撑体系

1.3.1 概览

大数据的核心价值是决策，但前提是必须有强大的技术支撑。纵观各种大数据解决方案，我们可以抽象出大数据系统的一般技术支撑体系。

一个完整的大数据系统总是由数据采集、数据存储、数据分析（或数据处理与服务）和数据应用四个部分构成的。图 1-1 给出了大数据系统的技术支撑体系。

大数据应用：监视、拦截、报告、推荐、可视化	系统 管理	服务 质量	数据 治理	数据 集成
大数据分析：识别、分析、模型				
大数据存储：获取、变动与整理、分布式存储				
大数据采集：数据源、数据位置、数据格式				

图 1-1 大数据的技术支撑体系

这个“大数据技术支撑体系”描述了大数据系统的最高抽象模式（架构）。可以看到，总体上，大数据系统的底层首先是大数据采集，其来源具有多样性；接着通过数据接口（如数据导入器、数据过滤、数据清洗、数据转换等）将数据存储于大规模分布式存储系统中；在数据存储的基础上，进一步实现数据分析（处理与服务），最终是大数据应用。

1.3.2 大数据采集层

本层考虑的第一个问题是数据来源。必须考虑所有渠道的、所有可用于分析的数据，这就要求公司或组织中的数据分析人员阐明执行分析所需要的各种类型数据。这正体现出大数据的第一特性，即巨量的信息资产。

技术上包含如下组件：

（1）企业遗留系统（Legacy System）。这些系统是公司或企业当前的应用程序，它们囊括了目前的主要数据。以下是一些典型的应用系统：客户关系管理系统、结算操作、大型机应用程序、企业资源规划、Web 应用程序、数据管理系统（DMS）、存储了逻辑数据流程策略和各种其他类型的文档，如 Microsoft Excel 电子表格、Microsoft Word 文档等。

（2）数据存储。数据存储包含企业数据仓库、操作数据库和事务数据库。这些数据通

常是结构化数据，可直接使用或轻松地转换来满足需求。这些数据不一定存储在分布式文件系统中，具体依赖于所处的环境。

(3) 智能设备。智能设备能够捕获、处理和传输使用最广泛的各类协议和格式的信息。这方面的例子包括智能仪表、摄像头、录音设备、医疗设备等，这些设备可用于执行各种类型的分析。绝大多数智能设备都会执行实时分析，但从智慧设备传来的信息也可批量分析。

(4) 聚合的数据提供程序。这些提供程序拥有或获取数据，并以复杂的格式和所需的频率通过特定的过滤器公开它。每天都会产生海量的数据，它们具有不同的格式，以不同的速度生成，而且通过各种数据提供程序、传感器和现有企业提供。

(5) 其他数据源。有许多数据来源于自动化的系统，地理信息（地图、地区详细信息、位置详细信息、人员详细信息、车辆详细信息），人类生成的内容（社交媒体、电子邮件、博客、微信、在线评论），传感器数据（环境：天气、降雨量、湿度；光电信息：电流、能源储能、导航装置、电离辐射、亚原子粒子等；物理：靠近、位置、角度、位移、距离、速度、加速度、震动、热量、热度、可见度、压力、流动、流体、速度、力、密度级别等）。

上述任何一种数据来源都需要依赖特定的技术。但无论技术复杂还是简单，核心问题是数据格式、数据类型（基本分为结构化、半结构化或非结构化三类）、数据率、数据量、数据源的位置（可能位于企业内或外部）、数据访问权限（访问权对数据的访问会影响可用于分析的数据范围）。

1.3.3 大数据存储层

这一层负责从数据源获取数据，并在必要时将它转换为适合数据分析方式的格式。例如，可能需要转换一幅图，才能将它存储在分布式文件系统或关系数据库管理系统（RDBMS）仓库中，以供进一步处理。注意，规范性制度和治理策略要求为不同的数据类型提供相适应的存储方式。

技术上，因为传入的数据可能具有不同的特征，所以数据变动（或转码）和存储层中的组件必须能够以各种频率、格式、大小和在各种通信渠道上读取数据。

(1) 数据获取。从各种数据源获取数据，并将其发送到数据整理组件或存储在指定的位置中。此组件必须足够智能或自动化（人工方式与大数据特性背道而驰），能够选择是否和在何处存储传入的数据。它必须能够确定数据在存储前是否应改动，或者数据是否可直接发送到业务分析层。

(2) 数据整理。负责将数据修改为需要的格式,以实现分析用途。此组件可拥有简单的转换逻辑或复杂的统计算法来转换源数据,分析引擎将会确定所需的特定数据格式。主要的挑战是容纳非结构化数据格式,如图像、音频、视频和其他二进制格式。

(3) 分布式数据存储。负责存储来自数据源的数据。通常,有多个数据存储选项,如分布式文件存储系统、云存储、非传统关系型数据库集群等。

1.3.4 大数据分析(处理与服务)层

分析层是大数据系统的核心和关键,体现大数据的价值特性。

分析层从存储层读取经过改动或整理后的数据。注意,在某些情况下,分析层可以直接从数据源访问数据。设计分析层需要认真地进行事先筹划和规划,必须确定如何管理以下任务的决策:生成想要的分析、从数据中获取洞察(observation),找到所需的实体、定位可提供这些实体的数据源、理解执行分析需要哪些算法和工具。

技术上包含如下组件:

(1) 分析层实体识别。负责识别和填充上下文实体,这是一个复杂的任务,需要高效的高性能流程管理系统支持。数据整理组件应为此实体识别组件提供补充,将数据修改为需要的格式。分析引擎将需要上下文实体来执行分析。

(2) 分析引擎。使用其他组件(具体来讲,包括实体鉴别、模型管理和分析算法)来处理和分析,分析引擎可能具有支持并行处理的各种不同的工作流、算法和工具。

(3) 模型管理。负责维护各种统计模型,验证和检验这些模型,通过持续培训模型来提高准确性。然后,模型管理组件会推广这些模型,它们可供实体识别或分析引擎组件使用。

1.3.5 大数据应用层

这一层使用了处理与服务层所提供的输出。使用者可以是可视化应用程序、人工、业务流程或服务。可视化分析层的结果可能具有挑战性,有时,看看类似市场中的竞争对手是如何做的会有所帮助。每一层包含多种组件类型,下面将会介绍这些不同类型的组件。

该层可用于检测欺诈,实时拦截交易,并将它们与使用已存储在公司或企业中的数据构建的视图进行关联。在欺诈性交易发生时,可以告知客户可能存在欺诈,以便及时采取更正操作。

此外,可以根据在数据改动层完成的分析来触发业务流程。可以启动自动化的步骤,

例如，如果客户接收了一条可自动触发的营销信息，则需要创建一个新订单，如果客户报告了欺诈，那么可以触发对信用卡使用的阻止。

分析的输出也可由推荐引擎使用，该引擎可将客户与他们喜欢的产品相匹配。推荐引擎分析可用的信息，并提供个性化且实时的推荐。

使用层还为内部用户提供理解、查询和导航企业内外的链锁信息的能力。对于内部使用者，为业务用户构建报告和仪表板的能力，使得利益相关者能够制定精明的决策并设计恰当的战略。为了提高操作有效性，可以从数据中生成实时业务警告，而且可以监视操作性的关键绩效指标。

技术上包含如下组件：

(1) 交易拦截器。此组件可实时拦截高容量交易，将它们转换为一种容易被分析层理解的实时格式，以便在传入数据上执行实时分析。事务拦截器应能够集成并处理来自各种来源的数据，如传感器、智能仪表、麦克风、摄像头、GPS 设备、ATM 和图像扫描仪。可以使用各种类型的适配器和 API 来连接到数据源；也可以使用各种加速器来简化开发，如实时优化和流分析，视频分析，银行、保险、零售、电信和公共运输领域的加速器，社交媒体分析，以及情绪分析器。

(2) 业务流程管理流程。来自分析层的洞察可供业务流程执行语言（BPEL）流程、API 或其他业务流程使用，通过自动化的上游和下游 IT 应用程序、人员和流程的功能，进一步获取业务价值。

(3) 实时监视。可以使用从分析中得出的数据来生成实时警告；可以将警告发送给感兴趣的使用者和设备，如智能手机和平板电脑；可以使用从分析组件生成的数据洞察，定义并监视关键绩效指标，以便确定操作有效性。实时数据可从各种来源以仪表板的形式向业务用户公开，以便监视系统的健康或度量营销活动的有效性。

(4) 报告引擎。生成与传统商业智能报告类似的报告的能力至关重要，用户可基于从分析层中得到的洞察，创建临时报告、计划的报告或自助查询和分析。

(5) 推荐引擎。基于来自分析层的分析结果，推荐引擎可向购物者提供实时的、相关的和个性化的推荐，提高电子商务交易中的转换率和每个订单的平均价值。该引擎实时处理可用信息并动态地响应每个用户，响应基于用户的实时活动、存储在 CRM 系统中的注册客户信息，以及非注册客户的社交概况。

(6) 可视化和发现。数据可跨企业内外的各种联邦的数据源进行导航。数据可能具有不同的内容和格式，所有数据（结构化、半结构化和非结构化）可组合起来进行可视化并

提供给决策者或任何需要的用户。此能力使得公司或组织能够将其传统的业务内容（包含在企业内容管理系统和数据仓库中）与新的社交内容（例如博客）组合到单个用户界面中。

1.3.6 垂直视图

影响逻辑层（大数据来源、数据改动和存储、分析和使用层）的所有组件的各方面都包含在垂直层中，它们分别是数据集成、数据治理、服务质量、系统管理。

（1）数据集成。大数据应用程序从各种数据起源、提供程序和数据源获取数据，并存储在 HDFS 或 NoSQL（Not only SQL，指非传统数据库）集群等数据存储系统中。这个垂直层可供各种组件使用（如数据获取、数据整理、模型管理和交易拦截器），负责连接到各种数据源。集成具有不同特征（如协议和连接性）的数据源信息，需要高质量的连接器和适配器。可以使用加速器连接到大多数已知和广泛使用的来源。这些加速器包括社交媒体适配器和天气数据适配器。各种组件还可以使用这一层在大数据存储中存储信息，从大数据存储中检索信息，以便处理这些信息。大多数大数据存储都提供了服务和 API 来存储和检索该信息。

（2）数据治理。数据治理涉及定义指南来帮助企业制定有关数据的正确决策。大数据治理有助于处理企业内或从外部来源传入的数据的复杂性、量和种类。在将数据传入企业进行处理、存储、分析和清除或归档时，需要强有力的指南和流程来监视、构建、存储和保护数据。

除了正常的数据治理考虑因素之外，大数据治理还包含其他因素：管理各种格式的大量数据、持续培训和管理必要的统计模型、以便对非结构化数据和分析进行预处理（注意，这是设置处理非结构化数据时的重要一步）、为外部数据设置有关其保留和使用的策略和规范性制度、定义数据归档和清除策略、创建如何跨各种系统复制数据的策略、设置数据加密策略。

（3）服务质量。该层定义数据质量、围绕隐私和安全性的策略、数据频率、每次抓取的数据大小和数据过滤器。

① 数据质量：完整地识别所有必要的元素，以可接受的新鲜度提供数据的时间轴，依照数据准确性规则来验证数据的准确性，采用一种通用语言（数据元组满足使用简单业务语言所表达的需求），依据数据一致性规则验证来自多个系统的数据一致性。在满足数据规范和信息架构指南基础上的技术规范性。

② 围绕隐私和安全的策略：需要策略来保护敏感数据。从外部机构和提供程序获取的

数据可能包含敏感数据（如社交媒体用户的联系信息或产品定价信息）。数据可以来源于不同的地区和国家，但必须进行相应的处理。必须制定有关数据屏蔽和这类数据的存储的决策。可以考虑以下数据访问策略：数据可用性、数据关键性、数据真实性、数据共享和发布、数据存储和保留（包括能否存储外部数据等问题）。如果能够存储数据，数据可存储多长时间？可存储何种类型的数据？数据提供程序约束（政策、技术和地区）、社交媒体使用条款。

③ 数据频率：提供新鲜数据的频率是多少？它是按需、连续还是离线的？

④ 抓取的数据大小：此属性有助于定义可抓取的数据，以及每次抓取后可使用的数据大小。

⑤ 数据过滤器：标准过滤器会删除不想要的数据和数据中的干扰数据，仅留下分析所需的数据。

（4）系统管理。系统管理对大数据至关重要，因为它涉及跨企业集群和边界的许多系统。对整个大数据生态系统的健康状况的监视包括：管理系统日志、虚拟机、应用程序和其他设备；关联各种日志，帮助调查和监视具体情形；监视实时警告和通知；使用显示各种参数的实时仪表板；引用有关系统的报告和详细分析；设定和遵守服务水平协议；管理存储和容量；归档和检索管理；执行系统恢复、集群管理、网络管理和策略管理。

1.4 大数据人才及其能力要求

大数据是时代的重要力量，而熟练掌握技术的人才根本。由于系统的复杂性和层次性，大数据人才也是由不同特点和不同能力要求的人才群组成的。

1.4.1 首席数据官

首席数据官（Chief Data Officer, CDO）和数据科学家（或称数据分析师）是企业大数据落地所需要人才的典型代表。

CDO 是懂得企业或组织业务运作的数据分析者，CDO 不能仅仅停留在简单收集、整理、分析、报告的层面，而是要结合实际，发现数据背后潜藏的挑战和机遇，并将挑战和机遇提交决策层，应用于企业战略规划和日常运营中。CDO 掌握企业内部核心的数据资源，对历史数据进行整理，对业务发展进行分析和预测。通过 CDO 改善企业在数据获取、存储和分析的水平，为高层管理者提供更科学有效的决策支持，以及开拓新的业务领域。CDO 领导数据分析团队，需要将数据转化为企业业务语言，从而使得决策层容易理

解和运用。通过 CDO 加强数据管控,可提高对业务风险的控制水平,是企业应对经济、金融危机的重要措施。

真正的 CDO 须具备五种能力:统计学、数学的能力,洞悉网络产业和发展趋势的能力,IT 设备和技术选型的能力,商业运营的能力,管理和沟通的能力。他们不仅要关注系统架构中所承载的内容,更要担任企业决策和数据分析汇整的枢纽;要熟悉面向服务的架构(SOA)、商业智能(BI)、大规模数据集成系统、数据存储交换机制,以及数据库、可扩展标记语言(XML)、电子数据交换(EDI)等系统架构;要深入了解企业的业务状况和所处的产业背景,清楚地了解组织的数据源、大小和结构等,才可将数据资料与业务状况联合起来分析,并提出相对应的市场和产品策略。

1.4.2 数据科学家(数据分析师)

与 CDO 一样,数据科学家是受到广泛关注的大数据专业人才。数据科学是一门交叉学科,涉及数学、统计学、计算机科学、数据可视化技术,以及具体行业的专业知识等。数据科学家的专长是“量化问题,然后解决问题”,他们的工作由三种内容混合而成:定量分析(让你了解数据),程序设计(让你可以处理数据),讲故事(让你了解数据的含义)。

数据科学家应该具有扎实的统计学基础,统计学是当前很多数据分析和数据挖掘算法的理论基础。对统计理论,如概率分布、假设检验、贝叶斯理论等的理解,有助于对数据进行更好的解读。

数据科学家应能够深刻理解预测模型,能够使用常见的预测模型,如回归、聚类、决策树等,在历史数据基础上预测未来。对这些预测模型使用方法、应用场景的理解,是数据科学家必须具备的技能。

数据科学家应当能够熟练使用统计工具。为了提高工作效率,数据科学家要熟练使用一种或多种分析工具。Excel 是当前最为流行的小规模数据处理工具,SAS 工具也在广泛应用。而以 Hadoop 为代表的管理工具,将越来越广泛地应用于数据业务中。

数据科学家基本、通用的一种技能是写代码,用所有相关方面都能听懂的语言进行沟通。另一种特殊技能是用数据讲故事,通过口头表达和视觉效果进行描述。数据科学家占支配地位的配置应是强烈的好奇心、想要深入问题内部的渴望,找到最核心的问题,提取成清晰的结论,并要经得起检验。

数据科学家是专业的数据研究者,承担对数据及其价值的专业性风险工作,需要熟练的数据处理和分析技能。CDO 职能更多涉及企业或组织总体管理和战略决策层面,数据科学家在 CDO 指导下工作,能解决复杂的数据问题但专业性更强。

一个初级的数据科学家可能只需要掌握基本分析技巧便可胜任；成熟的数据科学家需要对数据分析方法有较深入的理解；而优秀的数据科学家则应具备丰富的经验、广泛的知识面，能够独立设计和完成相关解决方案。总之，随着组织对数据的重视程度越来越高，数据科学家在经营和决策中所起的作用也越来越大，因而对数据科学家的技术技能和内在素质均提出了更多的要求。

成熟的数据分析师应具备四个条件：熟悉业务细节、掌握数据分析工具操作、对数据价值的敏感度和对数据提炼融合的能力。目前很多数据分析师比较擅长的是通过数据分析对已发生的问题查找原因，但缺乏发掘未知问题的能力，也缺少对趋势预测的把握，而大数据的价值恰恰在于预测未来。如果只熟悉数据分析工具操作，却不熟悉业务和运作细节，就无法从既有数据中挖掘出新的价值，达到推动企业发展的目的。

1.4.3 大数据开发工程师

BAT 三家互联网大公司对大数据开发工程师的招聘都希望是统计学和数学背景的硕士或博士学位，即要求有数学及统计学相关的背景。因为，缺乏理论背景的数据工作者，更容易进入一个技能上的危险区域。按照不同的数据模型和算法总能做出一些结果来，但如果不知道数据代表什么，也就不能得到真正有意义的结果，并且那样的结果还容易误导人。只有具备一定的理论知识，才能理解模型、复用模型甚至创新模型，来解决实际问题。

除了良好的数学背景外，对大数据开发工程师来说，还要求有很强的计算机编程能力。实际开发能力和大规模的数据处理能力是作为大数据开发工程师的一些必备要素。许多数据的价值来自于挖掘过程，开发人员必须亲自动手才能发现金子的价值。例如，现在人们在社交网络上所产生的许多记录都是非结构化的数据，如何从这些毫无头绪的文字、语音、图像甚至视频中攫取有意义的信息，就需要大数据开发工程师亲自挖掘。即使在某些团队中，大数据开发工程师的职责以商业分析为主，但也要熟悉计算机处理大数据的方式。

除了数学和统计学相关理论背景和超强的计算机编程能力外，作为大数据开发工程师还需要具有特定应用领域或行业的知识。大数据开发工程师这个角色很重要的一点是，不能脱离市场，因为大数据只有和特定领域的应用结合起来才能产生价值。他不能只是懂得数据，还要有商业头脑，不论对零售、医药、游戏还是旅游等行业，都能对其中某些领域有良好的理解，最好还与企业的业务方向一致。过去，我们常说一些奢侈品店员势利，看人一眼就知道买得起买不起，但这群人恰恰是有敏锐性的，他们是这个行业的专家。又如对医疗行业了解的人，他在考虑医疗保险业务时，不仅会查看人们去医院看病的相关记录，也会考虑饮食数据，这些都基于对该领域的了解。

最后, 对于一个优秀的大数据开发工程师来说, 除了上面列出的三点能力要求外, 还有另外一个非常重要的要求, 即他们必须深入理解大数据系统架构、各个组件的基本原理、实现机制甚至其中涉及的算法等。只有这样, 他们才能构建一个强大且稳定的分布式集群系统, 并充分利用其分布式存储和并行计算能力来处理大数据。

对于大多数企业而言, 由于自行研发一个高性能的集群系统往往要支付高昂的费用。经过多年的发展, 如今已形成了以 Hadoop 为核心的开源大数据生态系统, 利用通用的硬件就可以构建一个强大、稳定、简单并且高效的分布式集群系统计算系统, 就可以满足企业基础架构平台的需求, 付出相对低廉的代价就可以轻松处理超大规模的数据。因此, 大数据开发工程师必须深入理解以 Hadoop 为核心的大数据生态系统的系统架构, 原理及开发应用, 并具有充分的优化经验, 才能充分利用其处理超大规模数据, 甚至在 Hadoop 平台上开发特定应用的新组件。当然, 大数据开发工程师还需要具有大数据采集、大数据预处理、大数据存储与管理、分析挖掘与展现应用等大数据相关技术。

1.4.4 大数据运维工程师

企业除了大数据分析人才(开发人才)外, 还需要有数据的采集、管理、运维方面的人才。由于大数据系统是一个非常复杂的系统, 涉及的技术繁多, 尤其是在基于开源的平台下, 对大数据系统运维工程师提出了非常高的技术和能力要求。大数据系统运维工程师应熟悉 Java、Python、Shell 等语言; 熟悉 Hadoop 工作原理, 对 HDFS、MapReduce 运行过程还有深入理解, 有 MapReduce 开发经验, 熟悉数据仓库体系架构, 熟悉数据建模; 熟悉至少一种 RDBMS, 如 MySQL/Oracle/SQLServer, 熟练使用 SQL 语言, 懂 SQL 调优; 熟悉大数据生态圈及其他技术, 如 HBase、Storm、Spark、Impala、Kafka、Sqoop 等技术细节。

目前, 大数据系统运维方面的人才非常缺乏, 也很难培养。因为大数据系统是一个非常复杂的系统, 要想熟悉其中的每一个组件, 都是非常不容易的。这和其他如 MySQL、J2EE 等专业人才完全不能类比。所以, 企业要特别注意储备和培养大数据系统的运维人才。

1.5 本章小结

本章介绍了大数据兴起的时代背景, 分析了我国大数据的国家战略。本章重点介绍了大数据的概念和特性, 讨论了大数据的技术体系, 从层次结构上, 我们指出任何大数据系统都是由数据采集、存储、计算和应用四个部分构成的。最后我们阐述了大数据人才的特征与培养要求。

在整个 IT 界，大数据平台和组件并非只有 Hadoop 及其生态系统一个方面。但是，无论从学术还是应用上看，Hadoop 生态都是最成功的。本章主要对 Hadoop 生态系统及其组件技术进行概括性描述，为后续学习奠定概念基础。

2.1 Hadoop 生态系统

2.1.1 架构的基本理论

1. 概念

架构（Architecture）一词最初来源于建筑业，用于表示建筑物的整体结构模式和风格，它把整个建筑物看成一个系统，强调其整体受力空间结构方式的合理性和有效性。架构实际上是一个广泛应用的概念，通常指系统的整体结构及其组成部分的原则关系。

随着计算机、网络 and 软件技术的不断发展，信息系统变得日益复杂，因而架构一词在信息领域变得越来越重要。IEEE 给出的架构定义是：架构是一个系统的基础组织结构，包括系统的组件构成、组件之间的相互关系、系统和其所处环境的关系，以及指导系统设计和演化的相关准则。

必须指出，架构总是用来描述系统结构的，无论系统是指软件系统还是硬件系统，也无论是指业务系统还是应用系统，甚至不论是数据系统还是存储系统。由于信息系统可以从不同的角度去划分和观察，因此就出现了各种冠以“架构”的系统描述。例如，当我们

关注信息系统的软件构成方式时，就有软件架构的概念；如果关注系统的硬件构成方式，当然就有硬件架构（或物理架构）的概念；如果关注系统组件在处理数据过程的集中性与分散性，就有所谓集中式架构和分布式架构的概念。此外，我们知道，软件是指程序、数据和文档，所以软件架构可以进一步分为程序（或软件组件）架构、数据架构或数据库架构等概念。凡此种种，可见架构一词具有广泛的适用性。但不管架构一词如何被广泛使用，其核心理念都是强调系统的组织结构及其组成部分的关系准则。

2. 系统架构的设计目标

归根结底，人们之所以需要架构这个概念，是因为随着系统的复杂性越来越高，设计人员必须从系统的高度，对整个平台进行全局性的思考。

系统架构的设计目标主要包括可靠性、安全性、可伸缩性、可扩展性、可维护性等。

可靠性是指系统运行过程出错的概率非常小，因此用户的商业经营和管理完全可以依赖该系统。

安全性是指系统具有保护信息的能力，通过加密、认证、访问控制等手段确保系统的数据处于完整、可控的状态。

可伸缩性是指系统必须能够在用户的使用率、用户的数目增加很快的情况下，保持合理的性能，只有这样，才能适应用户的市场扩展的可能性。

可扩展性是指在新技术出现的时候，一个系统应当允许导入新技术，从而对现有系统进行功能和性能的扩展。

维护包括两方面，一是排除现有的错误，二是将新的需求反映到现有系统中去。一个易于维护的系统，意味着可以有效降低技术支持的花费。

3. 技术架构

技术架构是指信息系统中技术组件的组织结构，由于技术包括软件和硬件两个方面，因此技术架构也可以分为软件架构和硬件架构两个方面。

研究大数据的技术架构，有三个基本观点需要确立。

首先，对于大数据部署和应用来说，技术总是具体的，抽象地讨论技术没有太多意义。例如，商业银行中传统信息系统基本上采用了 IOE 技术，即以 IBM、Oracle 和 EMC 为代表的商业软件与相关硬件技术，而大数据时代，银行业则普遍在探讨所谓去 IOE 技术，也就是采用像 Hadoop 和 NoSQL（Not only SQL，指非传统关系型数据库）数据库这

样的开源或发布性开源技术。因此,分析和讨论大数据技术架构,应当而且必须以目前最重要的开源技术为对象。

其次,从应用的角度看,技术架构也是具体的,不同规模与性质的企业可能会有不同的系统架构,但既然是研究架构,则必然需要理解各种不同架构的共同点,以便揭示大数据系统设计的内在规律和演化趋势,从而指明正确的系统建设方向。

第三,大数据系统的最高抽象模式实际上就是大数据的基本系统架构,对这个顶层架构进行初次分解,就得到数据采集、数据存储、数据分析和数据应用四个方面,因此,技术架构也需要相应地从这四个方面描述。在涉及这些方面的时候,可能会出现各种角度的架构概念,如软件架构、总体架构、数据库架构、分布式架构、部署架构等,读者应当不难区分并理解它们的差异和核心。

2.1.2 主要组件及其关系

目前,以 Hadoop 为核心,整个大数据应用于研发已经形成了一个基本完善的生态系统。图 2-1 给出了 Hadoop 大数据应用生态中最主要的组件,该图描述了这些组件的地位,以及它们之间的相互作用关系。

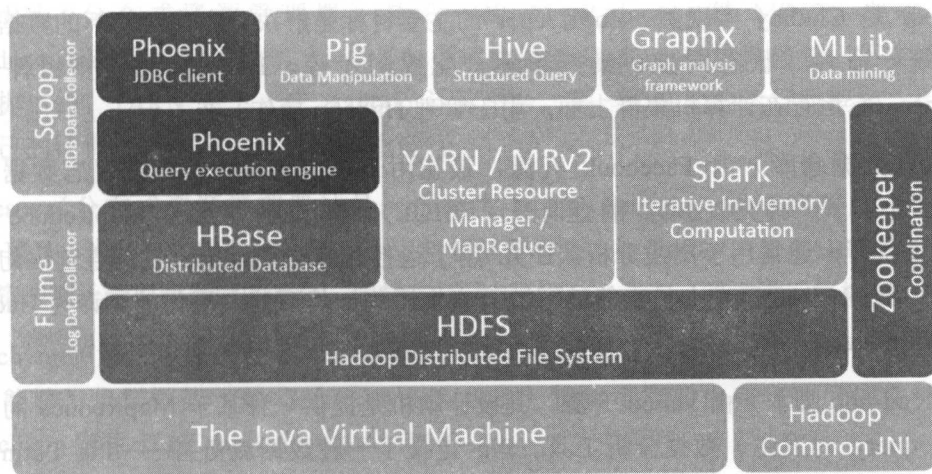


图 2-1 Hadoop 大数据应用生态中最主要的组件及其关系

HDFS (Hadoop 分布式文件系统) 源自于 Google 的 GFS 论文, 发表于 2003 年 10 月, HDFS 是 GFS 的实现版。HDFS 是 Hadoop 体系中数据存储管理的基础, 它是一个高度容错的系统, 能检测和应对硬件故障, 在低成本的通用硬件上运行。HDFS 简化了文件的一致性模型, 通过流式数据访问, 提供高吞吐量应用程序数据访问功能, 适合带有大型

数据集的应用程序。HDFS 提供一次写入多次读取的机制，数据以块的形式，同时分布存储在集群的不同物理机器上。

MapReduce（分布式计算框架）源自于 Google 的 MapReduce 论文，发表于 2004 年 12 月，Hadoop MapReduce 是 Google MapReduce 克隆版。MapReduce 是一种分布式计算模型，用以进行海量数据的计算。它屏蔽了分布式计算框架细节，将计算抽象成 Map 和 Reduce 两部分，其中 Map 对数据集上的独立元素进行指定的操作，生成键-值对形式中间结果。Reduce 则对中间结果中相同“键”的所有“值”进行规约，以得到最终结果。MapReduce 非常适合在大量计算机组成的分布式并行环境里进行数据处理。

HBase（分布式列存数据库）源自 Google 的 BigTable 论文，发表于 2006 年 11 月，HBase 是 Google BigTable 的实现。HBase 是一个建立在 HDFS 之上，面向结构化数据的可伸缩、高可靠、高性能、分布式和面向列的动态模式数据库。HBase 采用了 BigTable 的数据模型，即增强的稀疏排序映射表（Key/Value），其中，键由行关键字、列关键字和时间戳构成。HBase 提供了对大规模数据的随机、实时读写访问，同时，HBase 中保存的数据可以使用 MapReduce 来处理，它将数据存储和并行计算完美地结合在一起。

ZooKeeper（分布式协作服务）源自 Google 的 Chubby 论文，发表于 2006 年 11 月，ZooKeeper 是 Chubby 实现版。ZooKeeper 的主要目标是解决分布式环境下的数据管理问题，如统一命名、状态同步、集群管理、配置同步等。Hadoop 的许多组件依赖于 ZooKeeper，它运行在计算机集群上面，用于管理 Hadoop 操作。

Hive（数据仓库）由 Facebook 开源，最初用于解决海量结构化的日志数据统计问题。Hive 定义了一种类似 SQL 的查询语言（HQL），将 SQL 转化为 MapReduce 任务在 Hadoop 上执行，通常用于离线分析。HQL 用于运行存储在 Hadoop 上的查询语句，Hive 使不熟悉 MapReduce 开发人员也能编写数据查询语句，然后这些语句被翻译为 Hadoop 上面的 MapReduce 任务。

Pig（ad-hoc 脚本）由 yahoo 开源，其设计动机是提供一种基于 MapReduce 的 ad-hoc（计算在 query 时发生）数据分析工具。Pig 定义了一种数据流语言——Pig Latin，它是 MapReduce 编程的复杂性的抽象，Pig 平台包括运行环境和用于分析 Hadoop 数据集的脚本语言（Pig Latin）。其编译器将 Pig Latin 翻译成 MapReduce 程序序列，将脚本转换为 MapReduce 任务在 Hadoop 上执行，通常用于进行离线分析。

Sqoop（数据 ETL/同步工具）是 SQL-to-Hadoop 的缩写，主要用于传统数据库和 Hadoop 之间传输数据。数据的导入和导出本质上是 MapReduce 程序，充分利用了 MR 的

并行化和容错性。Sqoop 利用数据库技术描述数据架构，用于在关系数据库、数据仓库和 Hadoop 之间转移数据。

Flume（日志收集工具）是 Cloudera 开源的日志收集系统，具有分布式、高可靠、高容错、易于定制和扩展的特点。它将数据从产生、传输、处理并最终写入目标的路径的过程抽象为数据流，在具体的数据流中，数据源支持在 Flume 中定制数据发送方，从而支持收集各种不同协议数据。同时，Flume 数据流提供对日志数据进行简单处理的能力，如过滤、格式转换等。此外，Flume 还具有能够将日志写往各种数据目标（可定制）的能力。总的来说，Flume 是一个可扩展、适合复杂环境的海量日志收集系统，当然也可以用于收集其他类型数据

Mahout（数据挖掘算法库）起源于 2008 年，最初是 Apache Lucent 的子项目，它在极短的时间内取得了长足的发展，现在是 Apache 的顶级项目。Mahout 的主要目标是创建一些可扩展的机器学习领域经典算法的实现，旨在帮助开发人员更加方便、快捷地创建智能应用程序。Mahout 现在已经包含了聚类、分类、推荐引擎（协同过滤）和频繁集挖掘等广泛使用的数据挖掘方法。除了算法，Mahout 还包含数据的输入/输出工具、与其他存储系统（如数据库、MongoDB 或 Cassandra）集成的数据挖掘支持架构。

YARN（分布式资源管理器）是下一代 MapReduce，即 MRv2，是在第一代 MapReduce 基础上演变而来的，主要是为了解决原始 Hadoop 扩展性较差，不支持多计算框架而提出的。YARN 是下一代 Hadoop 计算平台，是一个通用的运行时框架，用户可以编写自己的计算框架，在该运行环境中运行。

Mesos（分布式资源管理器）是一个诞生于 UC Berkeley 的研究项目，现已成为 Apache 项目，当前有一些公司使用 Mesos 管理集群资源，如 Twitter。与 YARN 类似，Mesos 是一个资源统一管理和调度的平台，同样支持诸如 MR、steaming 等多种运算框架。

Tachyon（意为超光速粒子）是以内存为中心的分布式文件系统，拥有高性能和容错能力，能够为集群框架（如 Spark、MapReduce）提供可靠的内存级速度的文件共享服务。Tachyon 诞生于 UC Berkeley 的 AMPLab。

Spark（内存 DAG 计算模型）是一个 Apache 项目，被标榜为“快如闪电的集群计算”，它拥有一个繁荣的开源社区，并且是目前最活跃的 Apache 项目。最早 Spark 是 UC Berkeley AMP Lab 所开源的类 Hadoop MapReduce 的通用并行计算框架，Spark 提供了一个更快、更通用的数据处理平台。和 Hadoop 相比，Spark 可以让你的程序在内存中运行时速度提升 100 倍，或者在磁盘上运行时速度提升 10 倍。

Spark GraphX 最先是伯克利 AMP Lab 的一个分布式图计算框架项目，目前整合在 Spark 运行框架中，为其提供 BSP 大规模并行图计算能力。

Spark MLlib 是一个机器学习库，它提供了各种各样的算法，这些算法用来在集群上针对分类、回归、聚类、协同过滤等。

Kafka 是 Linkedin 于 2010 年 12 月开源的消息系统，主要用于处理活跃的流式数据。活跃的流式数据在 Web 网站应用中非常常见，这些数据包括网站的 PV (Page View)，用户访问了什么内容，搜索了什么内容等。这些数据通常以日志的形式记录下来，然后每隔一段时间进行一次统计处理。

Apache Phoenix 是 HBase 的 SQL 驱动 (HBase SQL 接口)，Phoenix 使得 HBase 支持通过 JDBC 的方式进行访问，并将你的 SQL 查询转换成 HBase 的扫描和相应的动作。

Apache Ambari 是安装部署配置管理工具，其作用就是创建、管理、监视 Hadoop 的集群，是为了让 Hadoop 以及相关的大数据软件更容易使用的一个 Web 工具。

在大数据的发展历程中，有大量不同类型和用途的组件相继产生，它们实现上述四个层次中的各种功能。但是，在激烈的竞争浪潮中，最终为业界和学术界认可并得到广泛应用的组件或框架只有大概 20 多种。下面我们将详细介绍其中的主要组件。

2.2 数 据 采 集

2.2.1 结构化数据采集工具

在 Hadoop 生态圈，Sqoop 作为 Apache 顶级项目，主要用来在 Hadoop 和关系数据库中传递数据。通过 Sqoop，可以方便地将数据从关系数据库导入 HDFS 或 HBASE 和 HIVE 中，或者将数据从 HDFS 导出到关系数据库。图 2-2 是 Sqoop 系统架构示意图。

Sqoop 架构非常简单，它主要通过 JDBC 和关系数据库进行交互。理论上讲，支持 JDBC 的数据库都可以使用 Sqoop 和 HDFS 进行数据交互。

Sqoop 数据导入具有以下特点：

- 支持文本文件、avrodatafile、SequenceFiles；
- 支持数据追加，通过 `append` 指定；
- 支持 `table` 列选取，支持数据选取 (`where`)，和 `table` 一起使用；

- 支持数据选取，如读入多表 join 后的数据，不可以和 table 同时使用；
- 支持 map 数定制；
- 支持压缩（compress）；
- 支持将关系数据库中的数据导入到 Hive（Hive-import）、HBase（hbase-table）。

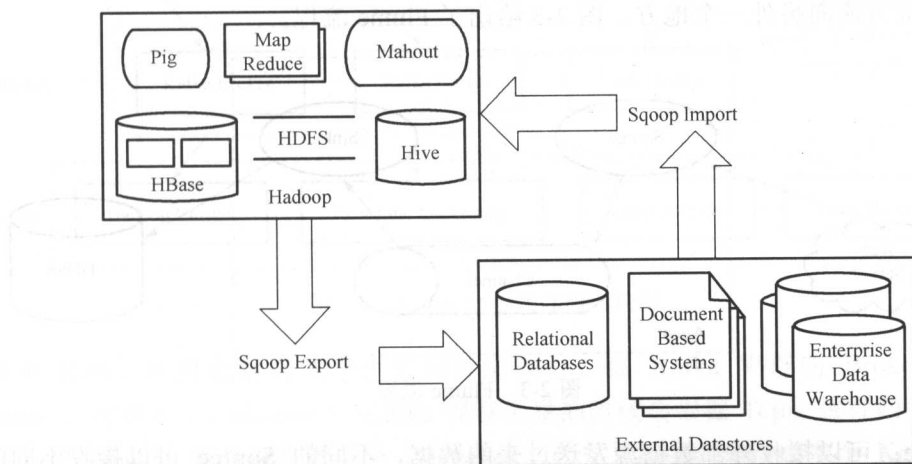


图 2-2 Sqoop 数据导入导出架构示意图

2.2.2 日志文件采集工具与技术

日志收集是大数据的基石，企业内部的业务平台每天都会产生大量的日志数据。收集组织的业务日志数据，供离线和在线的分析系统使用，是日志收集系统要做的事情。高可用性、高可靠性和可扩展性是日志收集系统需要具有的基本特征。

1. 日志收集

日志收集模块需要使用一个分布式的，具有可靠性、高可用性，能够处理海量日志数据的框架，并且应该能够支持多源采集和集中存储。目前常用的开源日志收集系统有 Apache Flume、Scribe 等。Flume 是由 Cloudera 开发的一个分布式、可靠和高可用的海量日志聚合系统，支持在系统中定制各类数据发送方，用于收集数据；同时，Flume 提供对数据进行简单处理，并写入各种数据接收方（可定制）的能力。

Flume 的工作流程是先收集数据源的数据，再将数据发送到接收方。为了保证这个过程的可可靠性，在送到接收方之前，会先对数据进行缓存，等到数据真正到达接收方后，才会删除自己之前缓存的数据。

Flume 传输的数据基本单位是事件（Event），如果是文本文件，通常是一行记录，这也

是事件的基本单位。事件 (Event) 从源 (Source) 传输到通道 (Channel)，再从通道传输到输出槽 (Sink)，事件本身用一个比特 (byte) 数组表示，并可携带消息头 (Headers) 信息。

Agent 是 Flume 的基本单位，一般情况下一个 Agent 由三个组件构成，分别是 Source、Channel 和 Sink。通过这三个组件，就可以完成整个数据收集工作，使 Event 能够从一个地方流向另外一个地方。图 2-3 给出了 Flume 流程。

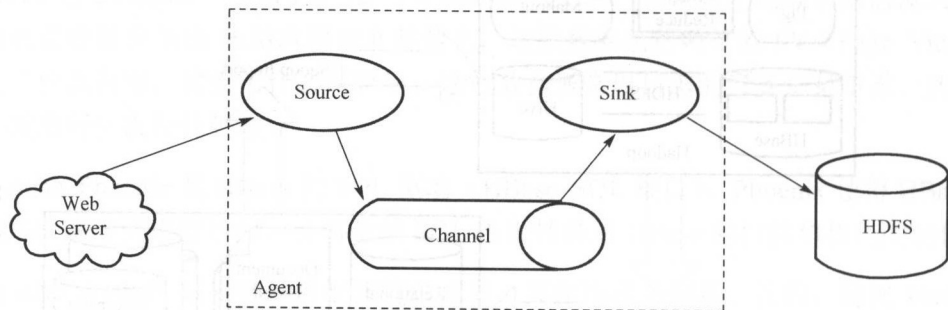


图 2-3 Flume 流程

Source: 可以接收外部数据源发送过来的数据，不同的 Source 可以接收不同的数据格式。比如 Spooling Directory，可以监视指定文件夹中的文件的变化，如果该目录中有新文件产生，就会立刻读取该文件中的内容。

Channel: 用来接收 Source 输出数据的缓存池，Channel 中的数据在进入到一个 Channel 或者进入终端时才会被删除，使得当 Agent 内部发生写入故障时不会造成数据的丢失，保证了数据收集的高可靠性。

Sink: 用于消费 Channel 中的数据，然后发送给数据接收方或者其他 Source，如数据可以写入到 HDFS 或者 HBase 中。

2. 数据分发工具 Kafka

Flume 收集的数据和进行日志处理的系统之间可能存在多对多的关系，为了解耦合保证数据的传输延迟，可以选用 Kafka 作为消息中间层进行日志中转分发。Flume 对于源数据流的发送速度不太稳定，有时快有时慢，而且当 Flume 的数据流发送速度过快时（这种情况很常见），会导致下游消费系统来不及处理，这样会使得有一部分数据可能会被丢弃。Kafka 在这两者之间可以扮演一个缓存的角色，而且数据是写入到磁盘上的，保证消息数据在系统正常启动/关闭时不会丢失。

Apache Kafka 是 Apache 软件基金会开发的一个开源的分布式消息订阅系统，该系统的设计目标是给实时数据处理提供一个统一、高吞吐、低等待的平台。Kafka 提供了实时

发布订阅的解决方案，克服了实时数据消费和比实时数据更大数量级的数据量增长的问题。Kafka 也支持 Hadoop 系统中的并行数据加载。图 2-4 是 Kafka 分布式订阅架构图。

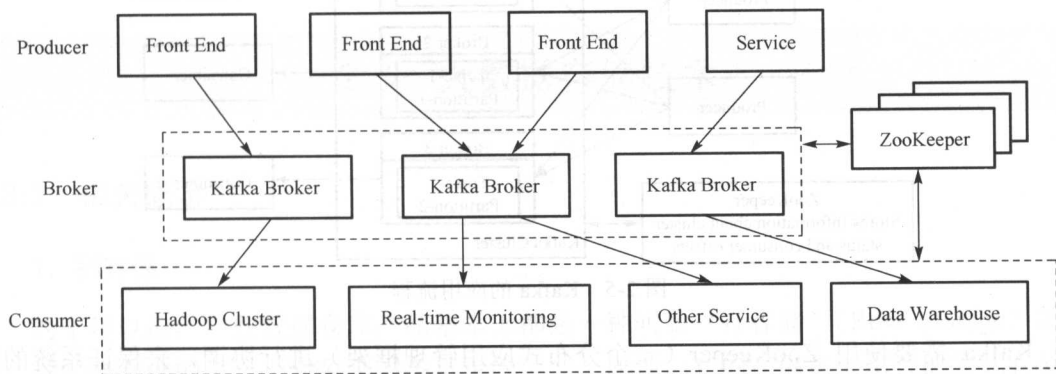


图 2-4 Kafka 分布式订阅架构图

整个 Kafka 集群包含有几个重要的组成部分，如 Kafka 集群的 Broker、生产者（Producer）、消费者（Consumer）。Kafka 保存消息的时候会根据 Topic 进行归类，发送消息者称为 Producer，消息接收者称为 Consumer。此外，Kafka 集群由多个 Kafka 实例组成，每个实例（Server）成为 Broker。无论是 Kafka 集群，还是 Producer 和 Consumer 都依赖于 ZooKeeper 来保证系统可用性集群保存一些 Meta 信息。

（1）Topic：是一个消息的基本单位。一个 Topic 可以看做一类消息，Kafka 进行消息保存的时候会按照 Topic 进行归类，每个 Topic 具体的数据存放位置由配置文件来决定，可能会存放不同的分区（Partition）上。每条消息在文件中的位置称为偏移量（Offset），偏移量是一个数据类型为 long 的数字，一对一地标记着每一条消息。

（2）Broker：是 Kafka 集群的基本单位，Kafka 集群中包含多个 Kafka 实例（Server），每个实例对应一个 Broker。集群中可能存在一个或多个代理（Broker）服务器，负责生产者和消费者二者之间的消息处理与交互。

（3）Producer：生产者，生产（发布）Topic 的进程。

（4）Consumer：消费者，消费（订阅）Topic 的进程，同时若干个消费者还可以组成一个消费组，这样当生产者发布 Topic 时可以实现两种常用功能。

- 只针对某一些 ID 或组 ID 对应的消费者通过点播发布消息；
- 通过广播将消息发给所有的消费者。

图 2-5 给出了 Kafka 的应用流程。

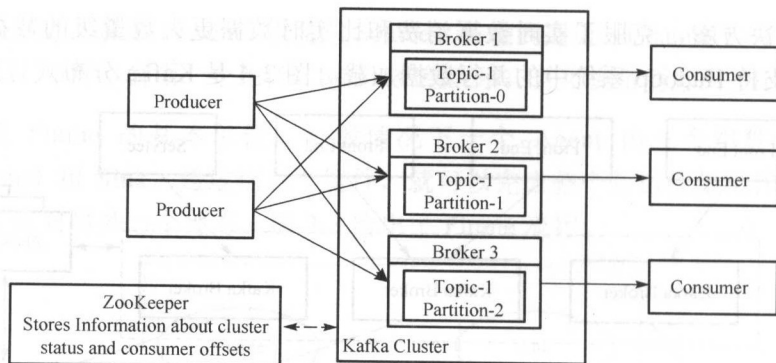


图 2-5 Kafka 的应用流程

Kafka 需要使用 ZooKeeper（一个分布式应用管理框架）进行协调，来保证系统的可用性，以及保存一些元数据（Meta Data）。ZooKeeper 与 Broker、Producer、Consumer 之间通过 TCP 协议进行通信。

Kafka 的典型使用场景如下。

（1）消息系统（Messaging）。对于一些常规的消息系统，Kafka 是个不错的选择。Partitions/Replication 和容错，可以使 Kafka 具有良好的扩展性和性能优势。不过，到目前为止，Kafka 还没有提供 JMS 中的事务性消息、传输担保（消息确认机制）、消息分组等企业级特性。Kafka 只能使用作为常规的消息系统，在一定程度上，尚未确保消息的发送与接收绝对可靠（如消息重发，消息发送丢失等）。

（2）网站活性跟踪（Websit Activity Tracking）。Kafka 可以作为“网站活性跟踪”的最佳工具，可以将网页/用户操作等信息发送到 Kafka 中，并实时监控，或者离线统计分析等。各种形式的 Web 活动会产生大量数据，用户活动事件包括登录、访问页面、单击链接，社交网络活动如喜欢、分享、评论，还有系统运作日志等。由于这些数据的高吞吐量（每秒百万级的消息），通常由日志记录系统和日志聚合解决方案来处理。这样的传统方案对提供日志数据给 Hadoop 这样的离线分析系统是可行的，但是，对于需要实时处理的系统就需要其他工具的支持了。

（3）日志收集中心（Log Aggregation）。Kafka 的特性决定它非常适合作为日志收集中心。Application 可以将操作日志批量、异步地发送到 Kafka 集群中，而不是保存在本地或者 DB 中。Kafka 可以批量提交消息/压缩消息等，这对 Producer 端而言，几乎感觉不到性能的开支。此时 Consumer 端可以使 Hadoop 等其他系统化的存储和分析系统。

总之，Kafka 是一个非常通用的系统，可以有許多 Producer 和很多的 Consumer 共享多个主题。相比之下，Flume 主要是被设计为往 HDFS、HBase 发送数据，它对 HDFS 有

特殊的优化，并且集成了 Hadoop 的安全特性。所以，如果数据被多个系统消费的话，Cloudera 建议使用 Kafka；如果数据被设计给 Hadoop 使用，使用 Flume。

2.3 大数据存储技术

2.3.1 相关概念

1. 列存储

约十年以前，一些数据仓库产品基本上都是一种叫做“行存储”（Row Storage）的产品，这是指存储的下一个对象是同条记录的下一个属性。它们在磁盘上用行的方式存储数据。SQL 服务器以前就是这样的。传统行存储结构如表 2-1 所示，它和关系型数据库 DB2、Oracle、Sybase、SQLServer、Greenplum、Netezza 和 Teradata 等都采用行存储模式。

表 2-1 传统的行存储数据排列方式

	Column 1	Column 2	Column 3	Column 4	Column 5
Row 1	Data 1-1	Data 1-2	Data 1-3	Data 1-4	Data 1-5
Row 2	Data 2-1	Data 2-2	Data 2-3	Data 2-4	Data 2-5
Row 3	Data 3-1	Data 3-2	Data 3-3	Data 3-4	Data 3-5
Row 4	Data 4-1	Data 4-2	Data 4-3	Data 4-4	Data 4-5

Vertica 的公司进行了颠覆性的改变，把行转 90°，变成列，用列的方式存储数据，如表 2-2 所示。

表 2-2 列存储数据排列

	Row 1	Row 2	Row 3	Row 4	Row 5
Column 1	Data 1-1	Data 2-1	Data 3-1	Data 4-1	Data 5-1
Column 2	Data 1-2	Data 2-2	Data 3-2	Data 4-2	Data 5-2
Column 3	Data 1-3	Data 2-3	Data 3-3	Data 4-3	Data 5-3
Column 4	Data 1-4	Data 2-4	Data 3-4	Data 4-4	Data 5-4
Column 5	Data 1-5	Data 2-5	Data 3-5	Data 4-5	Data 5-5

在列式存储模式下，存储空间中的下一个对象就从同一条记录的下一个属性转变为下一条记录的同属性。显然，这种旋转了 90° 的存储方式并没有减少数据量，但会带来这样一些好处：第一，大数据应用往往需要批量访问列数据（如果用户主要关心同一属性的统计特性），这时候列存储的优势就会体现出来，这种方式对于属性的访问比原来的行存储方式要快很多，据有关报道，它的读取速度比行存储产品要快 50~100 倍；第二，有利

于提高数据的压缩比,因为同类数据存储在一起有助于提高数据之间的相关性,从而有利于实施高效压缩算法(例如行程压缩算法等)。

两种存储的数据都是从上至下、从左向右的排列。行是列的组合,行存储以一行记录为单位,列存储以列数据集合单位,或称列族(Column Family)。行存储的读写过程是一致的,都是从第一列开始,到最后一列结束。列存储的读取是列数据集中的一段或者全部数据,写入时,一行记录被拆分为多列,每一列数据追加到对应列的末尾处。

但是,两种存储格式各自的特性都决定了它们都不可能是完美的解决方案。如果首要考虑是数据的完整性和可靠性,那么行存储是不二选择,列存储只有在增加磁盘并改进软件设计后才能接近这样的目标。如果以保存数据为主,行存储的写入性能比列存储高很多。在需要频繁读取单列集合数据的应用中,列存储是最合适的。如果每次读取多列,两个方案可酌情选择:采用行存储时,设计中应考虑减少或避免冗余列;若采用列存储方案,为保证读写效率,每列数据尽可能分别保存到不同的磁盘上,多个线程并行读写各自的数据,这样避免了磁盘竞用的同时也提高了处理效率。无论选择哪种方案,将相同属性的数据聚凑在一起都是必需的,这是减少磁头在磁盘上的移动,提高数据读取时间的有效办法。

在过去的十年里,正是由于这种颠覆性的转变,数据仓库产品的性能提升了 50 倍。表 2-3 给出了列式存储与行存储的比较。

表 2-3 行存储与列存储对比

比较对象	行 存 储	列 存 储
优点	写入效率高,提供数据完整性保证	读取过程没有冗余,适合数据定长的大数据计算
缺点	数据读取有冗余现象,影响计算速度	缺乏数据完整性保证,写入效率低
改进	优化存储格式,保证能够在内存快速删除冗余数据	多磁盘多线程并行写入/读(需要增加运营成本和修改软件)

目前,业界对两种存储方案有很多争持,焦点是谁能够更有效地处理海量数据,且兼顾安全、可靠、完整性。列存储由于以列为单位来存储数据,适合对某一列进行随机查询处理。采用列存储数据模型的数据库系统具有高扩展性,即使数据增加也不会降低相应的处理速度,因此,列存储主要适合应用于需要处理大量数据的情况。

在已知的几种大数据处理软件中,Hadoop 的 HBase 采用列存储,MongoDB 是文档型的行存储,Lexst 是二进制型的行存储,不适合用在 OLTP,或者更新操作,尤其是插入、删除操作频繁的场所。

2. Key-Value 存储

Google 在其分布式数据库技术产品 Bigtable 中,为存储 Web 页面,创造性地提出了 Key-Value 这种 MAP 数据结构,并广泛应用到 Google 的各种应用中。

键值 (Key-Value, KV) 存储数据库, 是一种 NoSQL (非关系型数据库) 模型, 其数据按照键值对的形式进行组织、索引和存储。KV 存储非常适合不涉及过多数据关系、业务关系的数据, 同时能有效减少读写磁盘的次数, 比 SQL 数据库存储拥有更好的读写性能。

这里, 我们以 Bigtable 为例, 介绍 Key-Value 数据结构。

Bigtable 采用 Key-Value 数据结构, Key 由行关键字、列关键字、时间戳组合而成, Value 为对应数据内容。行关键字和列关键字都是字符串数据类型; 时间戳是一个 64 位的长整数, 精确到毫秒; 这三个属性在一个数据库中全局唯一。由 Key 和 Value 构成的 KV 数据结构称为一个数据项。考虑到分布式数据库的多复本的特性, 数据项会按照时间戳进行排序, 并对于过期的数据项进行过期回收, 其数据结构如图 2-6 所示。

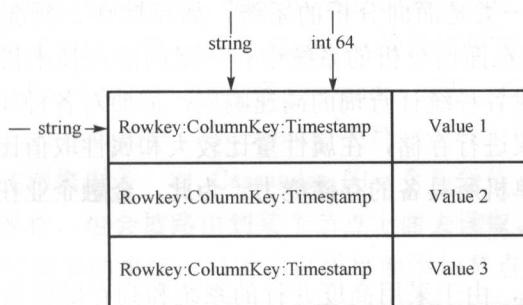


图 2-6 Bigtable KV 结构示意图

Key-Value 数据模型就是一个映射, Key 是查找数据地址的唯一关键字, 而 Value 数据则是实际存储的内容。它使用哈希函数实现关键字到值的快速映射, 这种模式可以提高数据的存储能力和并发读写能力, 适合通过主键快速查询。

目前有很多用于大数据处理的免费键值存储数据库, 例如 Memcached、Redis, Redis 是一个高性能的 Key-Value 存储系统, 和 Memcached 类似, 它支持存储的 Value 类型相对更多, 包括 String (字符串)、List (链表)、Set (集合) 和 Zset (有序集合)。与 Memcached 一样, 为了保证效率, 数据都是缓存在内存中, 区别的是 Redis 会周期性地把更新的数据写入磁盘或者把修改操作写入追加的记录文件, 并且在此基础上实现了主从同步。Redis 的出现, 很大程度补偿了 Memcached 这类 Key-Value 存储的不足, 在部分场合可以对关系数据库起到很好的补充作用。Redis 提供了 Python、Ruby、Erlang、PHP 客户端, 使用很方便。

3. NoSQL (Not only SQL) 数据库

随着以 NoSQL 为代表的新型数据管理软件日益成熟, 其在企业和组织信息管理系统中的应用也已逐步深入。

NoSQL 数据库，泛指非关系型的数据库，兴起的契机在于传统关系型数据库应对大规模、高并发的能力有限，而 NoSQL 的普遍性能优势能够弥补关系型数据库在这方面的不足。相比于传统数据库，以云平台为基础的 NoSQL 数据库系统具有以下特点：

- NoSQL 数据库去掉了传统关系数据库的关系特征，因此，很易于扩展；
- 由于 NoSQL 数据库结构简单，所以它在大数据量下的读写性能很好；
- NoSQL 数据库可以随时存储自定义的数据格式；
- NoSQL 数据库可以方便地实现高可用的架构。

这些特点使得 NoSQL 数据库系统在金融行业具有较大的应用研究前景。金融信息系统中主要包括两类的管理需求：一类是以事务处理为主的高一致性的系统需要，如银行的核心信息系统；另一类是面向分析的系统，如反欺诈、反洗钱、客户关系管理系统等。NoSQL 数据库系统在面向分析的系统中有一定的潜在技术优势，例如，在传统联机分析系统中，为了支撑对各种统计查询的高速响应，需要对各种属性组合预计算，这就需要对大量的中间计算结果进行存储，在属性量比较大和属性取值比较多的情况下，所需的计算资源往往大大超过单机所具备的存储能力。为此，金融企业在构建联机分析系统时需要投入大量资源。

NoSQL 数据库系统，由于采用高度并行的系统和列存储的数据管理结构，其数据查询具有弹性查扩展的特点，在响应大规模聚集查询时相对于传统的数据库系统结构具有较大的优势。因此，NoSQL 数据库系统已成为很多企业数据仓库系统建设的候选数据管理平台。随着 NoSQL 数据库系统的事务处理能力越来越强，其在企业和组织的应用范围将越来越广。

常见的几类 NoSQL 产品如下。

(1) Key-Value 存储 (Memcached、Redis)。这类 NoSQL 产品在互联网业内应用范围最广。Memcached 提供具备 LRU 淘汰策略的 KV 内存存储；而 Redis 提供支持复杂结构 (List、Hash 等) 的内存及持久化存储。Redis 适用于数据变化快且数据库大小可预见 (适合内存容量) 的应用程序，如股票价格、数据分析、实时数据搜集、实时通信。

(2) 列存储型 (HBase、Cassandra)。HBase 是基于列式存储的分布式数据库集群系统。列存储由于以列为单位来存储数据，适合对某一列进行随机查询处理。采用列存储数据模型的数据库系统具有高扩展性，即使数据增加也不会降低相应的处理速度，因此，列存储主要适合应用于需要处理大量数据的情况。

HBase 是 Apache Hadoop 生态系统中的重要一员，实现了对海量数据的随机实时读写

访问。从逻辑上讲, HBase 将数据按照表、行和列进行存储。HBase 的主要目标是依靠横向扩展, 通过不断增加廉价的商用服务器, 来增加计算和存储能力。HBase 提供了命令行管理和丰富的 API 访问接口, 通过这些接口调用, 可以使用多种程序语言对 HBase 进行访问。

HBase 适用于偏好 Bigtable, 并且需要对大数据进行随机、实时访问的场合, 如 Facebook 消息数据库。HBase 是一个写快读慢的系统(当然, 这里的慢是相对于写来说的)。若生产环境是一个 read heavy 场景, 可对 HBase 做读优化, 主要手段是增强系统 IO 能力(HDFS 层面)、增大 BlockCache、调整 Major Compaction 策略等。若随机读为主, 还可以调小 Blocksize。

Cassandra 也是面向列的开源分布式 NoSQL 数据库系统。它最初由 Facebook 开发, 用于存储收件箱等简单格式数据, 集 Google Bigtable 的数据模型与 Amazon Dynamo 的完全分布式的架构于一身。Facebook 于 2008 将 Cassandra 开源, Cassandra 具有分布式、基于 Column 的结构化及高伸展性特性。Cassandra 不是一个数据库, 而是由一堆数据库节点共同构成的一个分布式网络服务, 对 Cassandra 的一个写操作, 会被复制到其他节点上去, 对 Cassandra 的读操作, 也会被路由到某个节点上面去读取。对于一个 Cassandra 群集来说, 扩展性能是比较简单的事情, 只管在群集里面添加节点就可以了。Cassandra 是一个混合型的非关系的数据库, 类似于 Google 的 Bigtable, 它是一个网络社交云计算方面理想的数据库。使用 Cassandra, 像文档存储, 你不必提前解决记录中的字段。你可以在系统运行时随意的添加或移除字段。这是一个惊人的效率提升, 特别是在大型部署上。Cassandra 是纯粹意义上的水平扩展, 为给集群添加更多容量, 可以指向另一台电脑。不必重启任何进程, 改变应用查询, 或手动迁移任何数据。可以调整你的节点布局来避免某一个数据中心出问题, 一个备用的数据中心将至少有每条记录的完全复制。支持范围查询, 采用列表数据结构: 在混合模式可以将超级列添加到 5 维。对于每个用户的索引, 这是非常方便的, 采用分布式写操作, 有可以在任何地方任何时间集中读或写任何数据, 并且不会有任何单点失败。由于其无单点、无中心、适合在线业务, 目前, 使用 Cassandra 的公司主要有 360 等。

(3) 文档存储(MongoDB、CouchDB)。文档存储不需要定义表结构, 存储格式可以多样化, 适合存储非结构化数据。它可以通过复杂的查询条件来获取数据, 是很容易使用的 NoSQL 数据库。CouchDB 的最佳应用场景是: 适用于数据变化较少, 执行预定义查询, 进行数据统计的应用程序。适用于需要提供数据版本支持的应用程序, 如 CRM 和 CMS 系统。

MongoDB 的最佳应用场景是: 需要动态查询支持; 需要使用索引而不是 Map/Reduce

功能；需要对大数据库有性能要求；需要使用 CouchDB 但因为数据改变太频繁而占满内存的应用程序。

4. 图存储

图存储数据库基于图理论构建，使用节点、属性和边的概念。节点代表实体，属性保存与节点相关的信息，边用来表示实体之间的关系。图数据库存储某些数据集时速度非常快，可以把图直接映射到面向对象的应用程序中。例如，最庞大、最复杂的 Graph 模型是人的关系，理论上用图描述并且用 Graph 数据库存储最合适不过，不过目前的数据规模、系统性能仍然有待优化。

Web2.0 时代，NoSQL 产品在互联网行业中的重要性随着互联网及移动互联网的发展而与日俱增。在大型互联网应用中，为应对大规模、高并发访问，大多都引入了 NoSQL 产品，其中 Memcached、Redis 以其高成熟度、高性能、高稳定性而被广泛使用。例如，微博平台具备了千台规模的 NoSQL 集群，微博核心的 Feed 业务、关系业务也都依赖 Memcached 及 Redis 提供高性能服务。

2.3.2 分布式文件存储系统

在当前数据呈爆炸式增长的模式下，单台物理机无论从存储还是从计算能力上都不能满足人们的需求。云计算的一大优势就是能够快速、高效地处理海量数据。在数据爆炸的今天，这一点至关重要。为了保证数据的高可靠性，云计算通常会采用分布式存储技术，将数据存储在不同的物理设备中。这种模式不仅摆脱了硬件设备的限制，同时扩展性更好，能够快速响应用户需求的变化。

分布式存储与传统的网络存储并不完全一样，传统的网络存储系统采用集中的存储服务器存放所有数据，存储服务器成为系统性能的瓶颈，不能满足大规模存储应用的需要。分布式网络存储系统采用可扩展的系统结构，利用多台存储服务器分担存储负荷，利用位置服务器定位存储信息，它不但提高了系统的可靠性、可用性和存取效率，还易于扩展。在当前的云计算领域，Google 的 GFS 和 Hadoop 开发的开源系统 HDFS 是比较流行的两种云计算分布式存储系统。近年来，在大数据处理方面应用最广泛、发展最迅速的分布式架构就是 Hadoop。下面以 Hadoop 的 HDFS 为例，对海量数据的分布式存储进行介绍。

1. HDFS

Hadoop 的出现解决了传统的单机处理模式受到机器内存、计算能力限制的问题，利用集群的存储和计算能力为海量数据提供可靠的存储和处理。

Hadoop是由 Apache 基金会开发的一个开源的分布式系统基础架构,提供了一系列数据并行处理工具和应用解决方案,并且具有高度可伸缩性,可以根据数据规模和需求动态地增加或删除节点。用户可以在不了解其分布式底层细节的情况下,方便地在普通硬件上架设自己的大规模集群系统,开发分布式程序,从而充分地利用集群系统的能力进行高速运算和存储。

HDFS 设计思路有以下几点。

(1) 硬件错误是常态。在数据中心,硬件异常应被视为常态而非异常状态。在一个大数据环境下, HDFS 集群由大量物理机器构成,每台机器由很多硬件组成,因为某一个组件出错而出错的几率是很高的,因此 HDFS 架构的一个核心设计目标就是能够快速检测硬件失效并快速从失效中恢复工作。

(2) 流式访问要求。在 HDFS 集群上运行的应用要求流式访问数据, HDFS 设计为适用于批处理而非交互式处理,因此在架构设计时更加强调高吞吐量而非低延迟。

(3) 大数据集。假定 HDFS 的典型文件大小是 GB 甚至 TB 大小的, HDFS 设计重点是支持大文件,支持通过机器数量扩展以支持更大的集群,单个集群应提供海量文件数量支持。

(4) 简单一致性模型。HDFS 提供的访问模型是一次写入多次读取的模型。写入后文件保持原样不动简化了数据一致性模型,并且对应用来说,它能得到更高的吞吐量,支持文件追加。

(5) 移动计算比移动数据代价更低。HDFS 利用了计算机系统的数据本地化原理,认为数据离 CPU 越近,性能更高。HDFS 提供接口让应用感知数据的物理存储位置。

(6) 异构软硬件平台兼容。HDFS 被设计成能方便地从一个平台迁移到另外一个平台。

按如上思路设计的 HDFS,是 Hadoop 的可扩展的分布式文件系统,用于大型的、分布式的、对大量数据进行访问的文件系统。Hadoop 运行于廉价的普通硬件上,具有高容错性与高 I/O 吞吐量。大部分 ICT 厂商,包括 Yahoo、Intel 的云计划采用的都是 Hadoop 平台的 HDFS 的数据存储技术。

如图 2-7 所示, HDFS 集群由一个 Master (NameNode) 和多个 Slave (DataNode) 组成。

在 HDFS 内部,一个文件中的数据按照某种固定大小(如 128 MB)的块(Block)存储,每个块可以按照用户指定的副本量存储在不同的机器上。NameNode 维护系统的命名空间,包括文件到块的映射关系、访问日志等属性的存储、操作功能等所有的元数据。文

件的基础信息等存放在 NameNode 当中, 采用集中式存储方案。NameNode 定期通过心跳消息与每一个 DataNode 通信, 给 DataNode 传递指令并收集其状态。HDFS 在集群实现中只能有一个 NameNode, 但是可以设置一个备份的 Secondary NameNode, 来保证系统的可靠性、容错性。

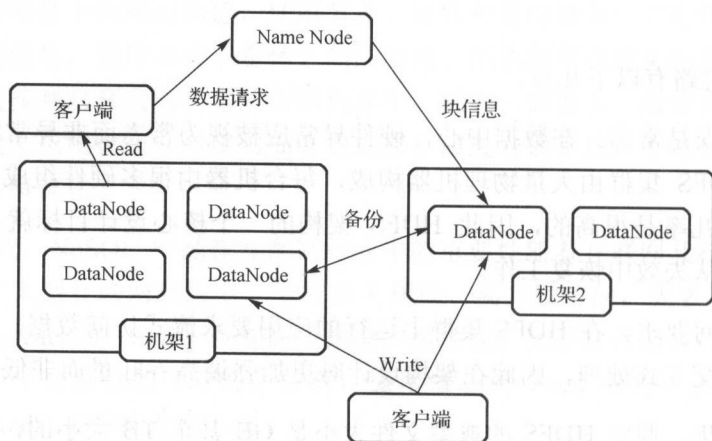


图 2-7 HDFS 架构

DataNode 提供文件内容的存储、操作功能。文件数据块本身存储在不同的 DataNode 中, DataNode 可以分布在不同机架。DataNode 定期与 NameNode 通信, 给 NameNode 发送状态并接收 NameNode 的指令。DataNode 启动之后会扫描本地文件系统中物理块个数, 并将对应的数据块信息汇报给 NameNode。

虽然 HDFS 是主从结构, 但客户端会分别访问 NameNode 和 DataNode 以获取文件的元信息及内容。HDFS 集群的客户端将直接访问 NameNode 和 DataNode, 相关数据直接从 NameNode 或者 DataNode 传送到客户端。

综合上述的设计假设和架构分析, HDFS 特别适合于以下场景:

- 顺序访问, 如提供流媒体服务等大文件存储场景。
- 大文件全量访问, 如要求对海量数据进行全量访问、OLAP 等。
- 整体预算有限, 想利用分布式计算的便利, 但又不打算购买昂贵的 HPC、高性能小型机等场景。

但是, HDFS 在如下场景其性能还是不尽如人意。

(1) 低延迟数据访问。低延迟数据访问意味着快速数据定位, 如 10 ms 级别响应, 系统若忙于响应此类要求, 则有悖于快速返回大量数据的假设。

(2) 大量小文件, 大量小文件将占用大量的文件块, 不仅会造成较大的浪费, 对元数据 (NameNode) 管理也是个严峻的挑战。Hadoop 的作业适用于较大的文件, 原因在于 Map 任务每次会处理一个输入的小文件 (FileInputFormat 通常是被分割的文件), 如果文件太小 (这里指的是小于 HDFS 的 Block 大小) 并且拥有很多这样的文件, 那么就会增加打开文件的性能开销。同时, 大量的小文件也会增加 NameNode 的元数据的存储开销。

(3) 多用户并发写入, 并发写入违背数据一致性模型, 数据可能不一致。

(4) 实时更新, HDFS 支持 Append, 实时更新会降低数据吞吐以及增加维护数据一致的代价。

2. 分布式内存文件存储 Tachyon

内存为王这句话现在很流行, 大数据处理对速度的追求是无止境的。由于内存的速度和磁盘的速度不是一个数量级, 同时, 内存的价格越来越低, 内存的容量也越来越大, 这就使得数据存在内存中有了可行性。伴随着这种趋势, 大量的基于内存的计算框架也出来了, 如 Spark, 就是优秀的基于内存的计算框架。但是, 现有的计算框架还面临一些挑战。Tachyon 的出现, 解决了内存计算存在的 GC 开销大、缓存数据丢失等问题。

Tachyon 是一个分布式内存文件系统, 可以在集群里以访问内存的速度来访问存在 Tachyon 里的文件。Tachyon 是安装在最底层的分布式文件存储和上层的各种计算框架之间的一种中间件, 主要职责是将那些不需要落地到 HDFS 里的文件, 存储到分布式内存文件系统中, 以此实现共享内存, 从而大幅提高访问效率。Tachyon 可以在不同框架内共享内存, 同时减少内存冗余和基于 JVM 内存计算框架的 GC 时间。

Tachyon 的架构是传统的 Master-slave 架构, 和 Hadoop 类似, TachyonMaster 里 WorkflowManager 是 Master 进程, 为了防止单点问题, 可以部署多台 Standby Master。Slave 是由 Worker Daemon 和 Ramdisk 构成。只有 Worker Daemon 是基于 JVM 的, Ramdisk 是一个 Off Heap Memory。Master 和 Worker 之间的通信协议是 Thrift。

图 2-8 表示了 Tachyon 的应用模式。

Tachyon 也有类似 RDD 的血统概念, 输入文件和输出文件都是会有血统关系, 这样来达到容错。同时, Tachyon 也利用血统关系, 异步实现检查点操作 (checkpoint)。在文件丢失情况下, 也能利用两种资源分配策略来优先计算丢失掉的资源。

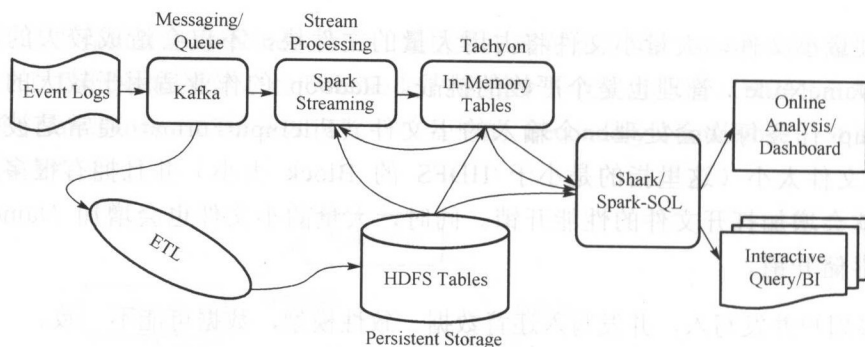


图 2-8 Tachyon 应用模式

2.3.3 数据库与数据仓库

1. HBase

HBase 即 Hadoop Database，是一个高可靠、高性能、面向列、可伸缩的分布式存储系统，利用 HBase 技术可在廉价 PC Server 上搭建起大规模结构化存储集群。

HBase 适合于存储大表数据（表的规模可以达到数十亿行以及数百万列），并且对大表数据的读、写访问可以达到实时级别。HBase 利用 Hadoop HDFS（Hadoop Distributed File System）作为其文件存储系统。在典型的大数据系统中，利用 Spark 和 HadoopMapReduce 来处理 HBase 中的海量数据，利用 ZooKeeper 作为协同服务。HBase 集群由主备 Master 进程和多个 RegionServer 进程组成，如图 2-9 所示。

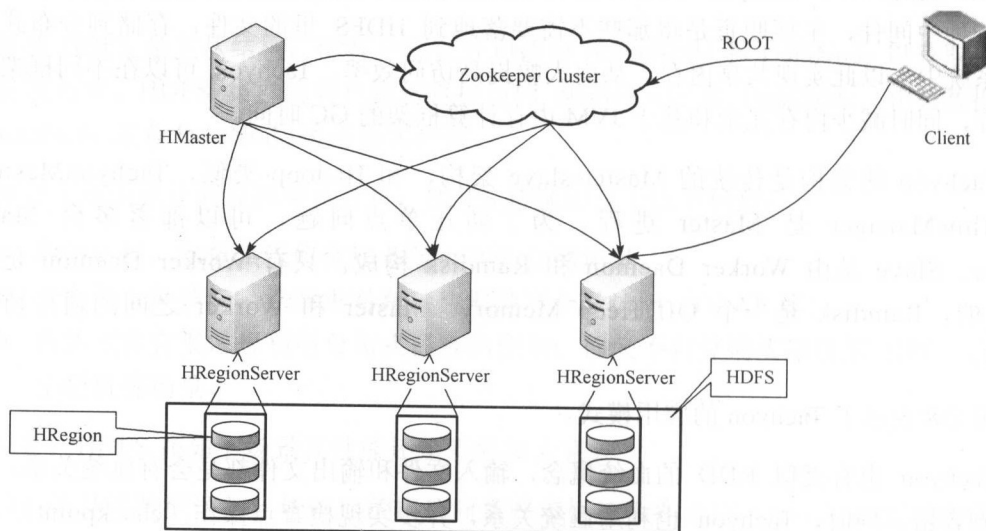


图 2-9 HBase 的系统结构

表 2-4 给出了 HBase 各组件的功能说明。

表 2-4 HBase 中各模块说明

组件名称	说 明
Client	Client 使用 HBase 的 RPC 机制与 Master、RegionServer 进行通信。Client 与 Master 进行管理类通信，与 RegionServer 进行数据操作类通信
Master	又叫 HMaster，在 HA 模式下，包含主用 Master 和备用 Master。主用 Master 负责 HBase 中 RegionServer 的管理，包括表的增删改查，RegionServer 的负载均衡，Region 分布调整，Region 分裂以及分裂后的 Region 分配，RegionServer 失效后的 Region 迁移等。当主用 Master 故障时，备用 Master 将取代主用 Master 对外提供服务，故障恢复后，原主用 Master 降为备用
RegionServer	RegionServer 负责提供表数据读写等服务，是 HBase 的数据处理和计算单元。RegionServer 一般与 HDFS 集群的 DataNode 合并，实现数据的存储功能
ZooKeeper 集群	ZooKeeper 为 HBase 集群中各进程提供分布式协作服务。各 RegionServer 将自己的信息注册到 Zookeeper 中，Master 据此感知各个 RegionServer 的健康状态
HDFS 集群	HDFS 为 HBase 提供高可靠的文件存储服务，HBase 的数据全部存储在 HDFS 中

(1) HBase 的查询过程。HBase 上的数据在 HDFS 是以 HFile 二进制的形式存储在 Block 中的，所以对于 HDFS 分布式文件系统来说，HBase 是完全透明的。HBase 数据访问流程如图 2-10 所示。

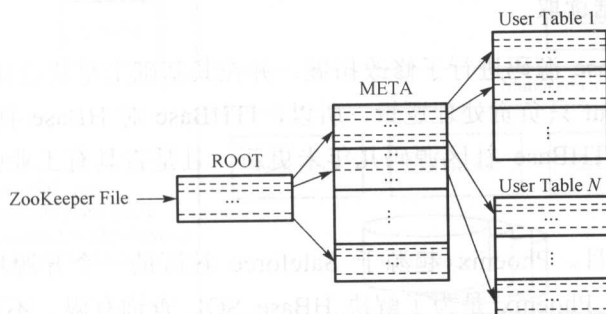


图 2-10 HBase 数据访问流程

HBase 的响应速度快是因为其特殊的存储模型和访问机制，HBase 中有两张表：META 表和 ROOT 表，META 表记录了用户的 Region 信息，包含了多个 Region 和所在的 RegionServer 的服务器地址，ROOT 表则记录了 META 表的 Region 信息。因此，ROOT 只有一个 Region。图 2-10 中，客户端可以快速定位到要查找的数据所在的 Region Server 服务器。当要对 HBase 进行增、删、改、查数据操作时，HBase Client 首先访问分布式协调服务 ZooKeeper，通过 ZooKeeper 可以访问到 ROOT 表的地址，因为 ROOT 表里面记录了 META 表的地址，接着通过 META 表就可以找到数据所在的位置，并将数据操作命令发送给 RegionServer，该 RegionServer 接收并执行该命令从而完成本次数据操作。

(2) 基于 HBase 的二级索引机制。HBase 具有扩展性强、实时查询效率高等特点，在大数据实时处理中应用十分广泛。但是，因为 HBase 的 Key-Value 存储特性，只支持少量的 SQL 查询操作，不支持二级索引。针对非主键查询，只能通过全表扫描和 Filter 过滤的方式获取数据，效率非常低，这使得 HBase 在许多查询的应用场景上存在很大的局限性。如果通过 Hive、Pig 等类似组件对全表进行 MapReduce 计算，依然会占用机器大量的资源，也会让应用的延迟大大增加。

如果可以在 HBase 存储的基础上，对表中一列 Value 做索引，而主键 RowKey 作为该索引的值，通过 Value 的索引快速定位到符合要求的 RowKey，再通过 RowKey 进行二次查找，将结果数据取出来，虽然这种方式会损失部分查询效率，但也能保证实时性，而且可以极大地提高查询的便捷性，这就是 HBase 二次索引机制的基本思想。针对以上问题，社区开发者贡献了许多架构方案来弥补 HBase 不支持二级索引的缺陷。

(3) ITHBase 拓展项目。ITHBase 是在 HBase0.19.3 版本中的第三方带索引的独立拓展项目，ITHBase 在 HBase 写入数据时，当 MemStore 写满后发出写磁盘的请求时，ITHBase 会拦截请求并为 MemStore 中的数据创建索引，索引会在表中以 CF 列的形式存在，而且 ITHBase 只支持 Region 级别的操作。当 ITHBase 读取数据时，会通过表中的索引列来加速 Scan 数据读取。

ITHBase 对 HBase 源码进行了修改拓展，并在其基础上重新设计实现了 HBase 里的 Server 模块，而 Client 只负责处理逻辑，所以，ITHBase 对 HBase 具有强侵入性。HBase 版本更新迅速，但 ITHBase 社区源码几年未更新，且是否具有工业强度的稳定性成为用户选择它的主要障碍。

(4) Phoenix 项目。Phoenix 起源于 Salesforce 社区的一个开源项目，后来发展成为 Apache 的顶级项目。Phoenix 是为了解决 HBase SQL 查询有限、不支持二级索引等问题开发的一个 Java 中间层，并提供可嵌入的 JDBC 驱动供客户端使用。通过发送 JDBC 请求给 HBase，协处理器将查询语言转化为多个 HBase Scan 操作和服务器端过滤，Phoenix 带来了更快的开发效率。

Phoenix 会将一个聚合查询分成多个 Scan，然后将 Scan 分配给 Phoenix 自定义的 HBase 协处理器，进行扫描操作并执行生成 JDBC 标准的查询结果。这些协处理器可以在服务器端并行工作来提高查询性能。平衡地拆分表是 Phoenix 能否获得高效查询的最重要因素之一，这包括将相等大小的分区平均分配到不同的 RegionServer 上，表中的数据在各个 RegionServer 上均匀分布可以保证每一个 Phoenix 线程处理的数据量相当，这样就可以减少查询的等待时间。

Phoenix 对于大数据集的查询可以达到毫秒级性能。当查询条件同时存在 RowKey 主键索引和二级索引时，会自动选择最优的索引。Phoenix 维护一个系统表作为 Scheme 元数据的存储，支持对多列进行动态索引的创建、删除和修改，不限制列数。当索引可变时，列数越多，写入速度受影响越大；索引不可变时，不影响写入速度，且 Phoenix 提供对 RowKey 分析的特性，可以让数据均匀分布在各个 Region。Phoenix 还具备其他值得关注的特性，例如，通过客户端批处理支持有限的事务，支持版本化的模式仓库，优化扫描等。目前，Phoenix 对 HBase 表现出完善的支撑，包括索引更新、增量识别等功能。

2. 数据仓库架构 Hive

Hive 是基于 HDFS 和 MapReduce 架构的数据仓库，提供类似 SQL 的 Hive QL 语言操作结构化数据，其基本原理是将 HQL 语言自动转换成 MapReduce 任务，从而完成对 Hadoop 集群中存储的海量数据进行查询和分析。图 2-11 是 Hive 的系统架构。

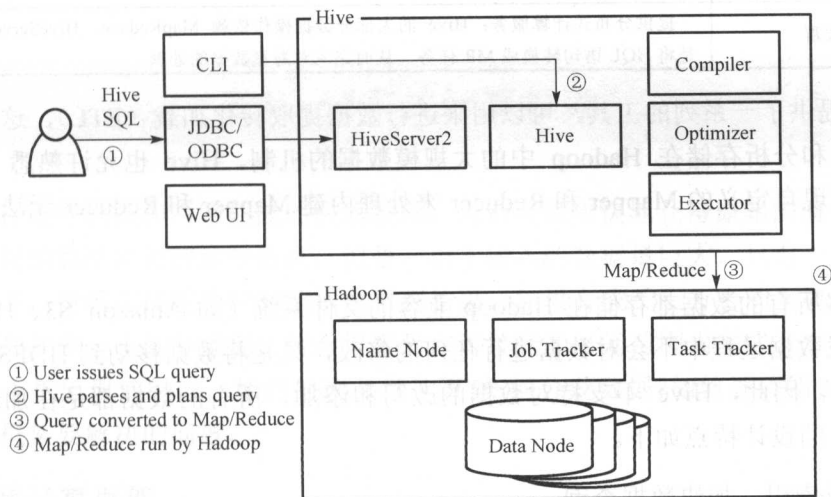


图 2-11 Hive 的系统架构

Hive 支持海量结构化数据分析汇总，将复杂的 MapReduce 编写任务简化为 SQL 语句，具有灵活的数据存储格式，支持 JSON、CSV、TEXTFILE、RCFILE、SEQUENCEFILE 几种存储格式。

Hive 利用 Hadoop HDFS 作为其文件存储系统。Hive 数据库中的所有数据文件都可以存储在 Hadoop HDFS 文件系统中，Hive 所有的数据操作也都是通过 Hadoop HDFS 接口进行的。

Hive 所有的数据计算都依赖于 MapReduce。对数据分析时，Hive 会将用户提交的 HQL 语句解析成相应的 MapReduce 任务并提交给 MapReduce 执行。

Hive 的 MetaStore（元数据服务）处理 Hive 的数据库、表、分区等结构和属性信息，这些信息需要存放在一个关系型数据库（如 MySQL）中，对 MetaStore 进行维护和处理。表 2-5 给出了 Hive 各个模块的说明。

表 2-5 Hive 模块说明

名 称	模 块 说 明
HiveServer	在 HA 模式下，包含主用 HiveServer 和备用 HiveServer。 主用 HiveServer：对外提供 Hive 数据库服务，将用户提交的 HQL 语句进行编译，解析成对应的 MR 任务或者 HDFS 操作，从而完成数据的提取、转换、分析。 备用 HiveServer：当主用 HiveServer 故障时，备用 HiveServer 将取代主用 HiveServer 对外提供服务
MetaStore	提供 Hive 的元数据服务，负责 Hive 表的结构和属性信息读、写、维护和修改
ZooKeeper 集群	ZooKeeper 为 HiveServer 的 HA 机制提供仲裁，各 HiveServer 将自己的信息注册到 ZooKeeper 中，为客户端访问主 HiveServer 提供依据
HDFS 集群	Hive 表数据存储在 HDFS 集群中
MR 集群	提供分布式计算服务：Hive 的大部分数据操作依赖 MapReduce，HiveServer 的主要功能是将 HQL 语句转换成 MR 任务，从而完成对海量数据的处理

Hive 提供了一系列的工具，可以用来进行数据提取转化加载（ETL），这是一种可以存储、查询和分析存储在 Hadoop 中的大规模数据的机制。Hive 也允许熟悉 MapReduce 的开发者实现自定义的 Mapper 和 Reducer 来处理内建 Mapper 和 Reducer 无法完成的复杂分析工作。

Hive 将所有数据都存储在 Hadoop 兼容的文件系统（如 Amazon S3、HDFS）中。Hive 在加载数据过程中不会对数据进行任何的修改，只是将数据移动到 HDFS 中 Hive 设定的目录下，因此，Hive 不支持对数据的改写和添加，所有的数据都是在加载的时候确定的。Hive 的设计特点如下。

- 支持索引，加快数据查询。
- 不同的存储类型，如纯文本文件、HBase 中的文件。
- 将元数据保存在关系数据库中，大大减少了在查询过程中执行语义检查的时间。
- 可以直接使用存储在 Hadoop 文件系统的数据。
- 内置大量用户函数（UDF）来操作时间、字符串和其他的数据挖掘工具，支持用户扩展 UDF 函数来完成内置函数无法实现的操作。
- 类 SQL 的查询方式，将 SQL 查询转换为 MapReduce 的工作在 Hadoop 集群上执行。Hive 与 SQL 相似促使其成为 Hadoop 与其他 BI 工具结合的理想交集。

由于 Hive 构建在基于静态批处理的 Hadoop 之上，Hadoop 通常都有较高的延迟并且在作业提交和调度的时候需要大量的开销，因此，Hive 并不能够在大规模数据集上实现

低延迟快速的查询。例如, Hive 在几百 MB 的数据集上执行查询一般有分钟级的时间延迟, 因此, Hive 并不适合那些需要低延迟的应用, 如联机事务处理 (OLTP)。Hive 查询操作过程严格遵守 Hadoop MapReduce 的作业执行模型, Hive 将用户的 HiveQL 语句通过解释器转换为 MapReduce 作业提交到 Hadoop 集群上, Hadoop 监控作业执行过程, 然后返回作业执行结果给用户。Hive 并非为联机事务处理而设计的, Hive 不提供实时的查询和基于行级的数据更新操作。Hive 的最佳使用场合是大数据集的批处理作业, 如网络日志分析。

根据行业务系统和大数据系统的功能划分, 大数据系统的存储至少要支持三种数据存储形式: 一是行式存储, 用于数据由传统数据库向大数据系统数据库过渡; 二是基于键值对的存储, 用于大体量、高并发数据的实时查询; 三是内存式数据库, 用于交互式数据分析和挖掘, 可通过构建分布式 Cube 加速性能, 也可部分使用 SSD 替代, 程序自动选择存储层。

2.4 分布式计算框架

随着各种硬件设备和网络性能的快速发展, 人们可以更快地获得原始的海量数据。虽然有很多专用的算法来处理这些数据, 但是, 由于输入的数据量巨大, 只有一台主机参与计算的情况下, 很难在接受的时间范围内得到结果。如果有成千上万台的主机参与计算, 每台主机处理其中一小部分数据, 就会很容易在可接受的时间范围内完成计算。但是当成千上万台的主机出现的时候, 如何处理并行计算? 如何分发数据? 如何处理错误? 这些都是分布式计算框架要解决的问题。

2.4.1 离线计算框架

1. MapReduce

Hadoop 架构的 MapReduce 是出现最早、知名度最大的分布式计算框架, 最早由 Google Lab 开发, 主要适用于大批量的集群任务。MapReduce 致力于解决大规模数据处理的问题, 在设计之初就考虑了数据的局部性原理。MapReduce 利用局部性原理将整个问题分而治之。MapReduce 指的是 Map (映射) 和 Reduce (归约) 两种函数思想, 也是计算过程的两个阶段。Map 函数对类型为 Key-Value 的数据进行处理, 之后交给 Reduce 函数进行归约处理。简单地说, MapReduce 就是将一个规模比较大的任务拆分成多个规模较小的任务同时并行进行作业, 各个规模较小的任务完成后, 再将它们归约聚合, 这也

就是并行分布式计算思想的体现。当一个任务失败时，调度器会挑选另一个节点重新开始作业，从而保证系统的可靠性、容错性。利用该模型，使用者只需关注想要执行的运算逻辑规则，而不必关心分布式执行中的容错、数据和计算分布、负载均衡等复杂的细节。MapReduce 框架会自动处理这些问题。图 2-12 给出了 MapReduce 的执行流程。

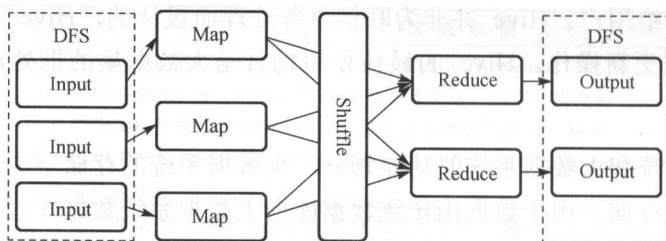


图 2-12 MapReduce 的执行流程

由于 Hadoop 相比传统计算模式的突出优势，基于 Hadoop 的应用已经被大量开发，尤其是在互联网领域，Hadoop 目前足以承担 PB 级别的大数据处理任务，其原生的 MapReduce 编程模型已经成为了主流的分布式编程模型。

MapReduce 集群由普通 PC 构成，是无共享式架构。在处理之前，将数据集分布至各个节点。处理时，每个节点就近读取本地存储的数据处理（Map），将处理后的数据进行合并（Combine）、排序（Shuffle and Sort）后再分发（至 Reduce 节点），避免了大量数据的传输，提高了处理效率。无共享式架构的另一个好处是配合复制（Replication）策略，集群可以具有良好的容错性，一部分节点的宕机对集群的正常工作不会造成影响。基于 MapReduce 的应用程序能够运行在由上千个商用机器组成的大型集群上，并以一种可靠容错的方式并行处理上 TB 级别的数据集。

MapReduce 适合 PB 级以上海量数据的离线处理。文件是 MapReduce 任务的数据的初始存储地。输入文件一般是存在 HDFS 里，这些文件的格式可以是任意的，可以是基于行的日志文件，也可以使用二进制格式，多行输入记录或其他一些格式。这些文件会很大——数十 GB 或更大。这种设计原理决定了其数据是静态的，不能动态变化，因此，MapReduce 不能处理流式计算，而且由于其是批量执行，因此时效性偏低。

MapReduce 模型的优点是灵活性很高，它不仅部署于 Hadoop 框架中，而且可以部署到多种分布式框架中。现在，云计算技术已逐渐成熟，Hadoop 可以轻松地部署在云上，而不会增加开销。

但是，MapReduce 的第一个版本（MRv1）也有缺点，其主要不足表现在大型集群上。当集群包含的节点超过 4000 个时（其中每个节点可能是多核的），就会表现出一定的

不可预测性, 其中一个最大的问题是级联故障, 由于要尝试复制数据和重载活动的节点, 所以一个故障会通过网络泛洪形式导致整个集群崩溃。总的说来, MRv1 存在扩展性受限、单点故障、难以支持 MP 之外的计算三方面的缺陷, 多计算框架各自为战, 数据共享困难。针对这些问题, YARN (MRv2) 产生了。

2. YARN (MRv2)

Hadoop 的发展并没有因为 MapReduce 框架的缺陷而停止。从 0.23.0 版本开始, Hadoop 开发团队摒弃了原有框架, 从根本上进行了改变。新的 Hadoop MapReduce 框架命名为 MapReduce v2 或者 YARN。

MRv1 的 JobTracker 和 TaskTracker 方法是一个重要的缺陷, 它关系到可伸缩性、资源利用和对与 MapReduce 不同的工作负载的支持。在 MapReduce 框架中, 作业执行受两种类型的进程控制: 一个称为 JobTracker 的主要进程, 它协调在集群上运行的所有作业, 分配要在 TaskTracker 上运行的 Map 和 Reduce 任务。许多被称为 TaskTracker 的下级进程运行分配的任务, 并定期向 JobTracker 报告进度。

大型的 Hadoop 集群出现了由单个 JobTracker 导致的可伸缩性瓶颈。例如, 在集群中有 5000 个节点和 40000 个任务同时运行时, 这样一种设计实际上就会受到限制。由于此限制, 不得不创建和维护更小的、功能更差的集群。此外, 较小和较大的 Hadoop 集群都从未最高效地使用它们的计算资源。在 Hadoop MapReduce 中, 每个从属节点上的计算资源由集群管理员分解为固定数量的 Map 和 Reduce Slot, 这些 Slot 不可替代。设定 Map Slot 和 Reduce Slot 的数量后, 节点在任何时刻都不能运行比 Map Slot 更多的 Map 任务, 即使没有 Reduce 任务在运行。这影响了集群的利用率, 因为在所有 Map Slot 都被使用 (而且我们还需要更多) 时, 我们无法使用任何 Reduce Slot, 即使它们可用; 反之亦然。

Hadoop 设计为仅运行 MapReduce 作业。随着替代性的编程模型 (如 Apache Giraph 所提供的图形处理) 的到来, 除 MapReduce 外, 越来越需要为可通过高效的、公平的方式在同一个集群上运行并共享资源的其他编程模型提供支持。

在 Hadoop MapReduce 中, JobTracker 具有两种不同的职责: ①管理集群中的计算资源, 这涉及维护活动节点列表、可用和占用的 Map 和 Reduce Slot 列表, 以及依据所选的调度策略将可用 Slot 分配给合适的作业和任务。②协调在集群上运行的所有任务, 涉及指导 TaskTracker 启动 Map 和 Reduce 任务, 监视任务的执行, 重新启动失败的任务, 推测性地运行缓慢的任务, 计算作业计数器值的总和, 等等。为单个进程安排大量职责会导致重大的可伸缩性问题, 尤其是在较大的集群上。

为了解决可伸缩性问题，一个简单而又绝妙的想法应运而生，这就是减少单个 JobTracker 的职责，将部分职责委派给 TaskTracker，因为集群中有许多 TaskTracker。在新设计中，这个概念通过将 JobTracker 的双重职责（集群资源管理和任务协调）分开为两种不同类型的进程来反映。

YARN 不再拥有单个 JobTracker，而是引入了一个集群管理器，它唯一的职责就是跟踪集群中的活动节点和可用资源，并将它们分配给任务。对于提交给集群的每个作业，会启动一个专用的、短暂的 JobTracker 来控制该作业中的任务的执行。短暂的 JobTracker 由在从属节点上运行的 TaskTracker 启动。因此，作业的生命周期的协调工作分散在集群中所有可用的机器上。得益于这种设计，更多工作可并行完成，可伸缩性得到了显著提高。

为了实现上述设计，在 YARN 中，将 MRv1 中的 JobTracker 的资源管理和任务调度/监控功能分离。在 YARN 框架中增加了一个 Resource Manager 来全局管理所有应用程序计算资源的分配，同时，每个节点中增加了一个 Node Manager，它是节点的代理。

YARN 在处理日志方面也做了较大的改进，现在的 Node Manager 会将日志进行压缩后传到文件系统中。

可以看出，旧的 Hadoop 架构受到了 JobTracker 的高度约束，JobTracker 负责整个集群的资源管理和作业调度。新的 YARN 架构打破了这种模型，允许一个新 Resource Manager 管理跨应用程序的资源使用，Application Master 负责管理作业的执行。这一更改消除了 MRv1 的瓶颈，还改善了将 Hadoop 集群扩展到比以前大得多的配置的能力。此外，不同于传统的 MapReduce，YARN 允许使用 Message Passing Interface 等标准通信模式，同时执行各种不同的编程模型，包括图形处理、迭代式处理、机器学习和一般集群计算。

随着 YARN 的出现，将不再受到简单的 MapReduce 开发模式约束，而是可以创建更复杂的分布式应用程序。实际上，可以将 MapReduce 模型视为 YARN 架构可运行的一些应用程序中的其中一个。这种能力非常强大，YARN 的使用模型几乎没有限制，不再需要与一个集群上可能存在的其他更复杂的分布式应用程序框架相隔离。随着 YARN 变得更加健全，它有能力取代其他一些分布式处理框架，从而完全消除专用于其他框架的资源开销，同时还简化了整个系统。

3. Spark

随着 YARN 的出现，MapReduce 使用者不需要担心任务的并行性和容错问题，只需要使用一些基本的操作就能并行地读写数据。但是，由于 MapReduce 框架并没有很好地使用分布式内存，每个 MapReduce 任务需要读写磁盘，这使得 MapReduce 对于

某些需要重用中间结果的应用很低效。在很多迭代式的机器学习和数据挖掘算法中，使用中间结果是非常常见的。如果使用 MapReduce 框架来处理这种类型的应用，那么数据副本、磁盘 I/O 和数据序列化将会花费大量时间。因为内存的读写速度远远高于磁盘，在理想状况下，如果所有的工作数据都能放入内存，那么大部分的任务就能在很短的时间内完成。

为了避免 MapReduce 框架中多次读写磁盘的消耗，更充分地利用内存，UC Berkeley AMP Lab 提出了一种新的开源的类 Hadoop MapReduce 的内存编程模型——Spark。Spark 基于 MapReduce 算法实现的分布式计算，拥有 Hadoop MapReduce 所具有的优点；但不同于 MapReduce 的是 Job 中间输出和结果可以保存在内存中，从而不再需要读写 HDFS，因此 Spark 能更好地适用于数据挖掘与机器学习等需要迭代的 MapReduce 的算法。Spark 掀内存计算之先河，以内存为赌注，引领大数据技术的发展。

Spark 是基于内存的分布式计算框架。在迭代计算的场景下，数据处理过程中的数据都会存储在内存中，从而避免了 MapReduce 计算框架中的问题。Spark 能够使用 Hadoop HDFS，使用户能够快速地从 MapReduce 切换到 Spark 计算平台上去；并且提供比 MapReduce 高 10~100 倍的性能。Spark 作为计算引擎，还支持小批量流式处理、离线批处理、SQL 查询、数据挖掘，避免用户在这几类不同的系统中加载同一份数据带来的存储 and 性能上的开销。Spark 能够融入 Hadoop 的生态系统，并且超越 Hadoop。

Spark 是用 Scala 编写而成的，Scala 是一种运行在 Java 虚拟机上的高级静态语言，是一种函数式编写语言。但 Spark 支持多语言编程，主要支持 Scala、Java、Python、R 四种编程语言，这一特点使得开发者可以在自己熟悉的语言环境下开发应用。Spark 自带了 80 多个内置算子，如 map、flatMap、groupBy、filter 等，这些丰富的 API 使得 Spark 易用性更强，同时允许在 Shell 中进行交互式计算。

目前，Spark 正在促使 Hadoop 和大数据生态系统发生演变，以便更好地支持大数据分析需求。Spark 已经在其核心的基础上发展了 Spark Streaming、Spark SQL、MLlib、GraphX 等功能模块，使其在支持传统批处理应用的同时，能够支持交互式查询、流计算、机器学习、图计算等各种应用，满足各种应用场景的需求。虽然这些功能各有不同，但是由于都是在 Spark 核心框架下提供支持的，使得我们可以简单地把以上各种功能整合到一起，而这在实际应用过程中具有很重要的意义。

Spark 也已经结合 Hadoop 等大数据处理工具发展出了自己的生态系统。Spark 的生态系统如图 2-13 所示，读者可以参考相关文献了解其中涉及的组件，本书将在后面的章节中重点介绍 Spark MLlib 的应用。

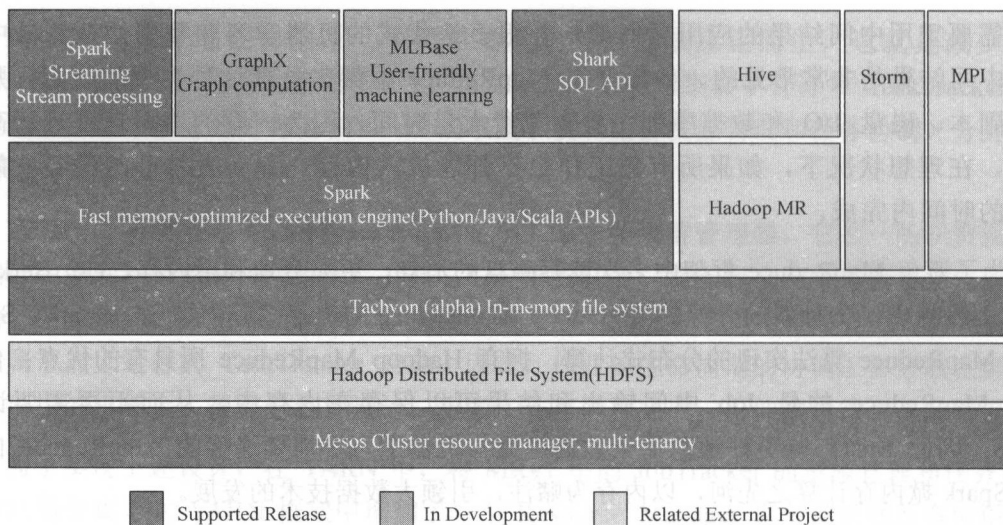


图 2-13 Spark 的生态系统

由于 Spark 是基于内存的迭代计算框架，因此非常适用于需要多次操作特定数据集的应用场合。需要反复操作的次数越多，所需读取的数据量越大，受益越大，数据量小但是计算密集度较大的场合，受益就相对较小。由于 RDD 的特性，Spark 不适合那种异步细粒度更新状态的应用，例如 Web 服务的存储或者增量的 Web 爬虫和索引，对于那种增量修改的应用模型不适合。

在学术界，Spark 早已得到各大院校的关注，Spark 源于加州大学伯克利分校，目前国内的一些知名大学也都开始对其展开相关的研究。而在工业界，Spark 也已经在互联网领域得到了广泛的应用。例如，国外 Cloudera、MapR 等大数据厂商全面支持 Spark 框架，Amazon 和 Yahoo 都在使用 Spark 进行日志分析。在国内，淘宝使用 Spark 进行用户交易数据分析，网易在使用 Spark 对海量数据进行报表和查询，腾讯使用 Spark 进行精准广告推荐等，豆瓣也在使用 Spark 的 python 克隆版 Dpark。可见 Spark 的优势之大与流行之广。

4. Flink

Flink 是一个高效、分布式、基于 Java 实现的通用大数据分析引擎，它具有分布式 MapReduce 一类平台的高效性、灵活性和扩展性，以及并行数据库查询优化方案，支持批量和基于流的数据分析，且提供了基于 Java 和 Scala 的 API。Flink 已升级成为 Apache 基金会的顶级项目，其设计思想主要来源于 Hadoop、MPP 数据库、流式计算系统等，支持增量迭代计算。

对 Flink 而言,其所要处理的主要场景就是流数据,批数据只是流数据的一个极限特例而已。换句话说,Flink 会把所有任务当成流来处理,这也是其最大的特点,Flink 可以支持毫秒级流式计算。

Flink 有几个最基础的概念,如 Client、JobManager 和 TaskManager。Client 用来提交任务给 JobManager,JobManager 分发任务给 TaskManager 去执行,然后 TaskManager 会心跳地汇报任务状态。看到这里,有的人应该已经有种回到上一代 Hadoop 的错觉。确实,从架构图去看,JobManager 很像当年的 JobTracker,TaskManager 也很像当年的 TaskTracker。然而有一个最重要的区别就是,TaskManager 之间是流(Stream)。其次,第一代 Hadoop 中,只有 Map 和 Reduce 之间的 Shuffle,而对 Flink 而言,可能是很多级,并且在 TaskManager 内部和 TaskManager 之间都会有数据传递,而不像 Hadoop,是固定的 Map 到 Reduce。

Flink 具有如下主要特征:

- DataSet API 支持 Java、Scala 和 Python;
- DataStream API 支持 Java 和 Scala;
- Table API 支持类 SQL;
- 有机器学习和图处理 (Gelly) 的各种库;
- 有自动优化迭代的功能,如有增量迭代;
- 支持高效序列化和反序列化。

Flink 与 Hadoop 兼容性很好。Flink 支持 Hadoop 所有的输入/输出格式和数据类型,这就使得开发者无须做任何修改就能够利用 Flink 运行历史遗留的 MapReduce 操作。

Flink 具有快速的特点。Flink 利用基于内存的数据流并将迭代处理算法深度集成到了系统的运行时中,这就使得系统能够以极快的速度来处理数据密集型和迭代任务。

Flink 可靠性和扩展性很高。当服务器内存被耗尽时,Flink 也能够很好地运行,这是因为 Flink 包含自己的内存管理组件、序列化框架和类型推理引擎。

Flink 的易用性也很好。在无须进行任何配置的情况下,Flink 内置的优化器就能够以最高效的方式在各种环境中执行程序。Flink 只需要三个命令就可以运行在 Hadoop 的新 MapReduce 框架 YARN 上。图 2-14 给出了 Flink on YARN 的运行机制。

Spark 和 Flink 都支持实时计算,且都可基于内存计算。而对于 Flink,其对于流式计算和迭代计算支持力度将会更加增强。无论是 Spark 还是 Flink,它们的发展重点都将是数据科学和平台 API 化,除了传统的统计算法外,还包括学习算法,同时使其生态系统越来越完善。

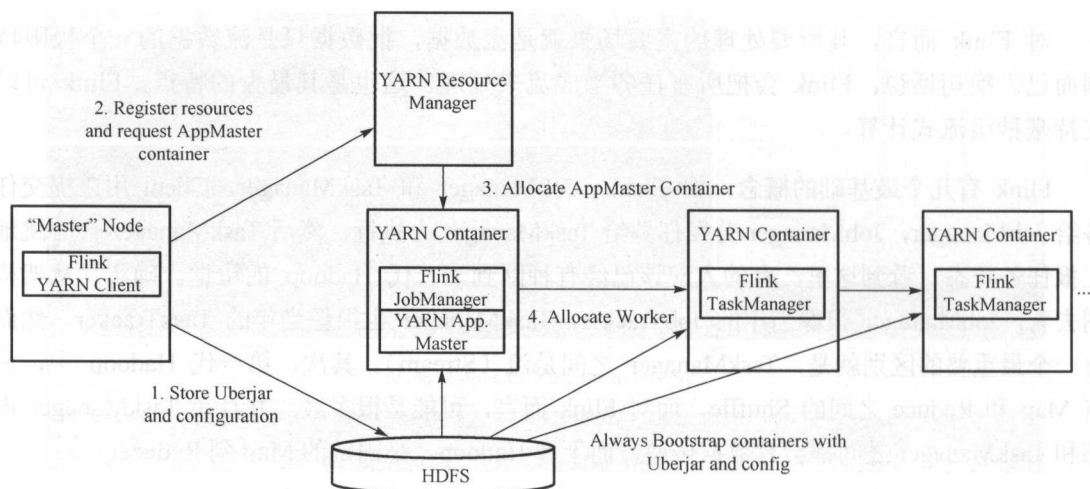


图 2-14 Flink on YARN 运行机制

2.4.2 实时流计算平台

在过去十多年中，随着互联网应用的高速发展，企业积累的数据量越来越大，越来越多。随着 Google Hadoop、MapReduce 等相关技术的出现，处理大规模数据变得简单起来，但是这些数据处理技术都不是实时的系统，它们的设计目标也不是实时计算。毕竟实时的计算系统和基于批处理模型的系统（如 Hadoop）有着本质的区别。

但是随着大数据业务的快速增长，针对大规模数据处理的实时计算变成了一种业务上的需求。例如，银行的实时营销和实时风险预警场景需要大数据平台具有历史数据快速统计、窗口时间内的信息流和触发事件及模型匹配、百毫秒级事件响应等性能。缺少实时的 Hadoop 系统已经成为整个大数据生态系统中的一个巨大缺失，Storm 正是在这样的需求背景下出现的。

在 Storm 出现之前，对于需要实现计算的任务，开发者需要手动维护一个消息队列和消息处理者所组成的实时处理网络，消息处理者从消息队列中取出消息进行处理，然后更新数据库，发送消息给其他队列。所有这些操作都需要开发者自己实现。

目前 Hadoop 平台通用的流处理引擎主要为 SparkStreaming 和 Storm，两者各有千秋，SparkStreaming 由时间窗口内批量事件流触发，Storm 由单个事件触发，单笔交易延迟方面 SparkStreaming 高于 Storm，但在整体吞吐量方面 SparkStreaming 略有提升。在进行 Hadoop 选型时主要考量处理引擎是否能够在流上实现统计类挖掘算法。

下面主要介绍实时编程模型 Storm 和 SparkStreaming。

1. Storm

Storm 是 Twitter 开源的一个类似于 Hadoop 的实时数据处理框架（原来是由 BackType 开发，后 BackType 被 Twitter 收购，将 Storm 作为 Twitter 的实时数据分析系统）。Storm 能处理高频数据和大规模数据的实时流计算，解决方案将被应用于实时搜索、高频交易和社交网络等。金融机构的交易系统是一个非常典型的流计算处理系统，它对系统的实时性和一致性有很高要求。

(1) Storm 的设计思想。在 Storm 中也有对于流 (Stream) 的抽象，如图 2-15 所示。Storm 则将流中元素抽象为 Tuple (元组)，一个 Tuple 就是一个值列表 (Value List)，List 中的每个 Value 都有一个 Name，并且该 Value 可以是基本类型、字符类型、字节数组等，当然也可以其他可序列化的类型。

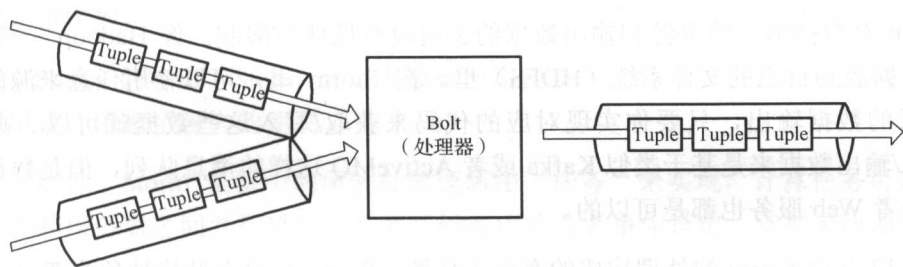


图 2-15 Storm 流

本质上，流是一个不间断的无界的连续 Tuple，Storm 在建模事件流时，把流中的事件抽象为 Tuple 即元组。Storm 认为每个 Stream 都有一个 Stream 源，也就是原始元组的源头，所以它将这个源头抽象为 Spout，Spout 可能是连接 Twitter API 并不断发出 Tweets，也可能是从某个队列中不断读取队列元素并装配为 Tuple 发射。

有了源头即 Spout 也就是有了 Stream，那么该如何处理 Stream 内的 Tuple 呢？同样，Twitter 将流的中间状态转换抽象为 Bolt。Bolt 可以消费任意数量的输入流，只要将流方向导向该 Bolt，同时它也可以发送新的流给其他 Bolt 使用，这样一来，只要打开特定的 Spout（管口）再将 Spout 中流出的 Tuple 导向特定的 Bolt，又 Bolt 对导入的流做处理后再导向其他 Bolt 或者目的地。

形象地说，Spout 就好像是一个一个的水龙头，并且每个水龙头里流出的水是不同的，我们想拿到哪种水就拧开哪个水龙头，然后使用管道将水龙头的水导向到一个水处理器 (Bolt)，水处理器处理后再使用管道导向另一个处理器或者存入容器中，如图 2-16 所示。

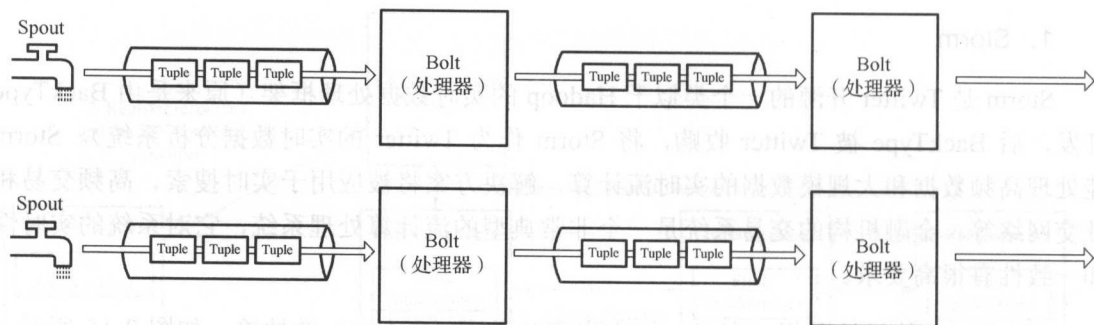


图 2-16 Spout 流的概念

为了增大水处理效率，很自然就想到在同一个水源处接上多个水龙头并使用多个水处理器，这样就可以提高效率。

Storm 对数据输入的来源和输出数据的去向没有做任何限制。像 Hadoop 一样，Storm 需要把数据放到自己的文件系统（HDFS）里。在 Storm 里，可以使用任意来源的数据输入和任意的数据输出，只要你实现对应的代码来获取/写入这些数据就可以。典型场景下，输入/输出数据来是基于类似 Kafka 或者 ActiveMQ 这样的消息队列，但是数据库、文件系统或者 Web 服务也都是可以的。

图 2-17 是由 Storm 的处理构成的有向无环图，Storm 将这个结构抽象为 Topology（拓扑）。拓扑是 Storm 最高层次的一个抽象概念，它可以被提交到 Storm 集群执行，一个拓扑就是一个流转换图，图中每个节点是一个 Spout 或者 Bolt，图中的边表示 Bolt 订阅了哪些流，当 Spout 或者 Bolt 发送元组到流时，它就发送元组到每个订阅了该流的 Bolt（这就意味着不需要我们手工拉管道，只要预先订阅，Spout 就会将流发到适当的 Bolt 上）。

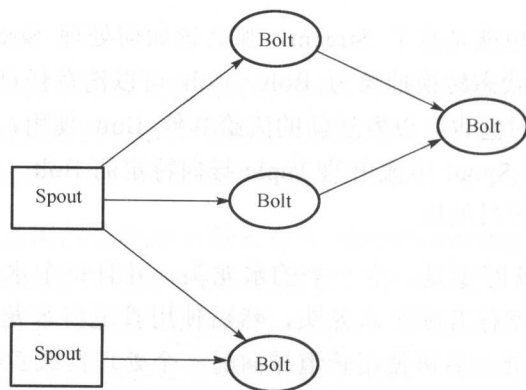


图 2-17 Storm 的拓扑

为了做实时计算，用户需要设计一个 Topology（拓扑图），并实现其中的 Bolt 处理细节，Storm 中拓扑定义仅仅是一些 Thrift 结构体，这样一来我们就可以使用其他语言来创建和提交拓扑。

拓扑的每个节点都要说明它所发射出的元组的字段的 Name，其他节点只需要订阅该 Name 就可以接收处理。

Storm 对数据的处理效率很高，而且支持水平扩展，具有高容错性。Storm 将数据流上的查询抽象为一个拓扑结构，它是一个由多个算子组成的有向无环图，主要有消息发送者和消息处理者两类算子。消息发送者既可以通过程序员编写的计逻辑自行生成数据，也可以接收外界的流数据；数据处理者既可以接收到的数据进行处理，也可以单纯地作为一个数据的传递者。数据在拓扑结构中被抽象成一个个的数据元组，数据流则被抽象成一组数据元组序列。

（2）Storm 的特点。Storm 针对大数据应用提供了一些原语供开发者使用，这为开发者的应用设计带来了很大的便利。

可扩展性：Storm 集群中的处理逻辑由“任务”来实现，计算任务可以在多个线程、进程和服务器之间并行进行，其并行程度由开发者事先设定，支持灵活的水平扩展。

高可靠性：Storm 的高可靠性表现在保证由消息发送者发出的每条消息都能被“完全处理”，这也是 Storm 直接区别于其他实时系统的地方。当消息被消息发送者发送出去的时候，系统会启动一个后台进程监控这个消息，并向消息发送者反馈。把消息经过各个节点的过程抽象成一棵消息树，如果某一个消息未被正常处理或者在规定的阈值之内未被处理，监控进程就会通知消息发送者重新发送该消息，直到这棵消息树中的消息都被完全处理了之后，消息发送者才会认为消息闭塞被安全处理。

高容错性：Storm 保证一个处理单元永远运行，除非使用者显式杀死这个处理单元。当一个处理单元出现运行故障时，Storm 会重新布置这个处理单元，不过，Storm 不负责保存各个处理单元的中间状态信息。如果某个处理单元中存储了中间状态，那么需要由开发者自行编写相关功能模块，以保障它在被重新启动时能够恢复故障前的状态。

支持多种编程语言：Storm 拥有一项多语言协议来支持多种编程语言。

（3）Storm 的应用。与 Hadoop 不同，Storm 是没有包括任何存储概念的计算系统，这就让 Storm 可以用在多种不同的场景下：非传统场景下数据动态到达或者数据存储在数据库这样的存储系统里，或数据是被实时操控其他设备的控制器（如交易系统）所消费。

Storm 有很多应用,如实时分析、在线机器学习 (Online Machine Learning)、连续计算 (Continuous Computation)、分布式远程过程调用 (RPC)、ETL 等。Storm 处理速度很快,每个节点每秒可以处理超过百万的数据组。它可扩展 (Scalable)、容错 (Fault-Tolerant),保证你的数据会被处理,并且很容易搭建和操作。

Twitter 产生趋势信息是 Storm 的一个典型应用。Twitter 从海量推文中抽取趋势信息,并在本地区域和国家层级进行维护。这意味着一旦一个案例开始出现,Twitter 的话题趋势算法就能实时地鉴别出这个话题。这个实时的算法就是通过在 Storm 上连续分析 Twitter 数据来实现的。

Storm 被广泛用来进行实时日志处理,出现在实时统计、实时风控,实时推荐等场景中。淘宝用它对超大量的日志进行统计,从中提取有用的信息。日志以持久的消息队列被读到 Spouts,然后在 Topologies 中,计算处理得到结果,最终结果被保存在数据库中供使用。淘宝使用 Storm 和消息队列结合,每天能够处理 2 百万到 15 亿条日志,日志量达到 2 TB 的近实时处理,这对实时处理大数据集合持续存储都有很高的要求。

2. Spark Streaming

Spark Streaming 是大规模流式数据并行处理框架,它将流式计算分解成一系列短小的批处理作业。Spark Streaming 是在 2013 年被添加到 Apache Spark 中的,作为核心 Spark API 的扩展,它允许数据工程师和数据科学家实时地处理来自于 Kafka、Flume,以及 Amazon Kinesis 等多种源的数据。Spark Streaming 提供了一套高效、可容错的准实时大规模流式处理框架,它能和批处理、即时查询放在同一个软件栈中,这种对不同数据的统一处理能力正是 Spark Streaming 会被大家迅速采用的关键原因之一。在 Spark Streaming 的用户群中,Uber、Netflix 和 Pinterest 等家喻户晓的公司都赫然在列。

(1) Spark Streaming 构架。计算流程: Spark Streaming 将流式计算分解成一系列短小的批处理作业,这里的批处理引擎是 Spark,也就是把 Spark Streaming 的输入数据按照 Batch Size (如 1 s) 分成一段一段的数据 (Discretized Stream), 每一段数据都转换成 Spark 中的 RDD (Resilient Distributed Dataset), 然后将 Spark Streaming 中对 DStream 的 Transformation 操作变为针对 Spark 中对 RDD 的 Transformation 操作,将 RDD 经过操作变成中间结果保存在内存中。整个流式计算根据业务的需求可以对中间的结果进行叠加,或者存储到外部设备。图 2-18 显示了 Spark Streaming 的整个流程。

(2) Spark Streaming 的应用。

① 网站流量统计。在互联网应用中,网站流量统计作为一种常用的应用模式,需要

在不同粒度上对不同数据进行统计,既有实时性的需求,又需要涉及聚合、去重、连接等较为复杂的统计需求。传统上,若使用 Hadoop MapReduce 框架,虽然可以容易地实现较为复杂的统计需求,但实时性却无法得到保证;反之若是采用 Storm 这样的流式框架,实时性虽可以得到保证,但需求的实现复杂度也大大提高了。Spark Streaming 在两者之间找到了一个平衡点,能够以准实时的方式容易地实现较为复杂的统计需求。

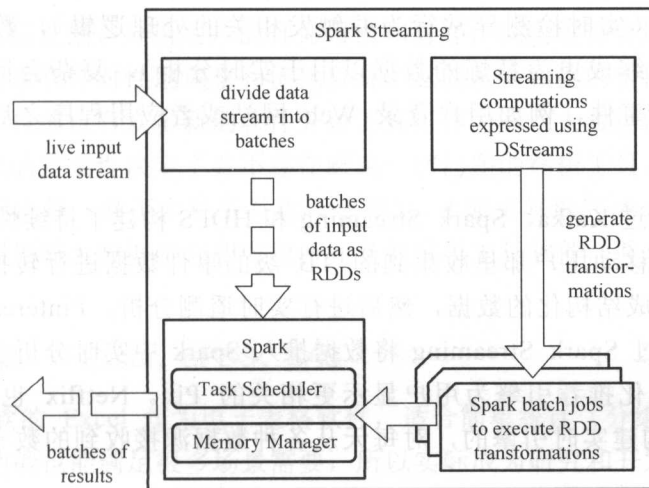


图 2-18 Spark Streaming 流程

② 数据暂存。Kafka 作为分布式消息队列,既有非常优秀的吞吐量,又有较高的可靠性和扩展性,在这里采用 Kafka 作为日志传递中间件来接收日志,抓取客户端发送的流量日志,同时接收 Spark Streaming 的请求,将流量日志按序发送给 Spark Streaming 集群。

③ 数据处理。将 Spark Streaming 集群与 Kafka 集群对接,Spark Streaming 从 Kafka 集群中获取流量日志并进行处理。Spark Streaming 会实时地从 Kafka 集群中获取数据并将其存储在内部的可用内存空间中。当每一个 Batch 窗口到来时,便对这些数据进行处理。

④ 结果存储。为了便于前端展示和页面请求,处理得到的结果将写入到数据库中。

相比于传统的处理框架,Kafka+Spark Streaming 的架构有以下几个优点。

- Spark 框架的高效和低延迟保证了 Spark Streaming 操作的准实时性。
- 利用 Spark 框架提供的丰富 API 和高灵活性,可以精简地写出较为复杂的算法。
- 编程模型的高度一致使得上手 Spark Streaming 相当容易,同时也可以保证业务逻辑在实时处理和批处理上的复用。

在基于 Kafka+Spark Streaming 的流量统计应用运行过程中，有时会遇到内存不足、GC 阻塞等各种问题。下面介绍一下如何对 Spark Streaming 应用程序进行调优来减少甚至避免这些问题的影响。

⑤ 从 Uber 到 Pinterest。虽然针对不同的目标和业务案例使用 Spark Streaming 的方式也不同，但其主要场景包括：流 ETL（将数据推入存储系统之前对其进行清洗和聚合），触发器（实时检测异常行为并触发相关的处理逻辑），数据浓缩（将实时数据与静态数据浓缩成更为精炼的数据以用于实时分析），复杂会话和持续学习（将与实时会话相关的事件，例如用户登录 Web 网站或者应用程序之后的行为，组合起来进行分析）。

例如，Uber 通过 Kafka、Spark Streaming 和 HDFS 构建了持续性的 ETL 管道，该管道首先对每天从移动用户那里收集到的 TB 级的事件数据进行转换，将原始的非结构化事件数据转换成结构化的数据，然后进行实时遥测分析。Pinterest 的 ETL 数据管道始于 Kafka，通过 Spark Streaming 将数据推入 Spark 中实时分析全球用户对 Pin 的使用情况，从而优化推荐引擎为用户显示更相关的 Pin。Netflix 也是通过 Kafka 和 Spark Streaming 构建实时引擎的，对每天从各种数据源接收到的数十亿事件进行分析完成电影推荐。

（3）Spark Streaming 与 Storm 的比较。Storm 和 Spark Streaming 在处理领域上有一定的差异，Storm 比较擅长实时性较高的数据处理，而 Spark streaming 则偏向于内存处理（注意严格来说内存处理跟流式处理并不是完全一样的）。

在数据安全性方面，Storm 虽然没有 Spark streaming 强大，但从目前来看已经够用了，要知道很多业务数据在这种场合下是允许丢失部分数据的。相对来说，在流式处理上，Storm 有着不可比拟的优势，而在大批量内存处理的方面，Spark Streaming 则占着不小的优势。但由于 Databricks 的强势推广加上 Spark 本身的快速更新，Spark 受到极大关注，非常火爆，而 Storm 的模型决定了它的应用范围相对要比 Spark 小很多（除了少数大公司，真正的实时流处理的需求并不多，而像 BAT 这样的公司又倾向于自己开发，不使用原生的架构），但 Storm 的好处在于 Storm 像 Hadoop 的 MapReduce 一样定义好了一套实时流处理的标准接口，大多数实时计算系统（包括 Twitter 新推出的 Heron）都遵循 Topology-Spout-Bolt 这一模式，这个体系是很有价值的。

总之，Spark Streaming 和 Storm 两者各有千秋，Spark Streaming 由时间窗口内批量事件流触发，Storm 由单个事件触发，单笔交易延迟方面 Spark Streaming 高于 Storm，但在整体吞吐量方面 Spark Streaming 略有提升。

2.5 数据分析平台与工具

2.5.1 面向大数据的数据挖掘与分析工具

从庞杂的数据背后挖掘、分析用户的行为习惯和喜好，找出更符合用户口味的产品和服务，并结合用户需求有针对性地调整和优化自身，这就是大数据的价值。但是，工欲善其事，必先利其器。目前已有众多新的软件分析工具正作为深入大数据洞察研究的重要手段，但是，现实情况的复杂性决定了并不存在解决一切问题的终极工具。开发过程中，需要根据实际情况灵活选择（甚至多种工具组合使用），才能更好地完成大数据挖掘和分析的任务。本节将介绍当前大数据研究和应用所涉及的常用工具软件，并进一步阐述其应用特点和适用场景。

1. 基础分析工具——Excel、SPSS、SAS

(1) Excel。微软的 Excel 作为电子表格软件，适合简单统计（分组/求和等）需求，由于其方便好用，功能也能满足很多场景需要，所以实际成为研究和开发人员最常用的数据处理软件工具。其缺点在于功能单一，且可处理数据规模小。近几年 Excel 在大数据方面（如地理可视化和网络关系分析）上也作出了一些增强，但应用能力仍然有限。

(2) SPSS。SPSS Statistics 作为商业统计软件，提供经典的统计分析（如回归、方差、因子、多变量等分析）处理。SPSS 是轻量级的，易于使用，但功能相对较少，适合常规统计分析。

(3) SAS。SAS 也是商业统计软件，功能丰富而强大（包括绘图能力），且支持编程扩展其分析能力，适合复杂与高要求的统计性分析，目前，有些企业特别是银行选择使用 SAS 进行大数据挖掘分析。

然而，上述三个软件在面对大数据环境时出现了各种不适。当然，这并不代表它们没有使用价值。在使用传统研究方法分析大数据时，如果海量原始数据经过预处理（如降维和统计汇总等）得到的中间规模不大，就很适合使用它们进行深入研究。

2. 基于机器学习的分析工具——SPSS Modeler、MATLAB、Weka

数据挖掘作为大数据应用的重要领域，在传统统计分析基础上，更强调提供机器学习的方法，关注高维空间下复杂数据关联关系和推演能力。代表是 SPSS Modeler（注意不是 SPSS Statistics，其前身为 Clementine），SPSS Modeler 的统计功能相对有限，主要是提

供面向商业挖掘的机器学习算法（决策树、神经网络、分类、聚类 and 预测等）的实现，同时，其数据预处理和结果辅助分析方面也相当方便，适合商业环境下的快速挖掘。不过，其处理能力难以应对亿级以上的数据规模。

另一个商业软件 MATLAB 也能提供大量数据挖掘的算法，但其特性更关注科学与工程计算领域。而著名的开源数据挖掘软件 Weka，功能较少，且数据预处理和结果分析也比较麻烦，更适合学术界或有数据预处理能力的用户。

3. 基于可视化的大数据分析工具——TableAU、Gephi、NanoCubes

近两年来出现了许多面向大数据、具备可视化能力的分析工具，在商业研究领域，TableAU 就是卓越代表。TableAU 的优势主要在于支持多种大数据源和格式，众多的可视化图表类型，加上拖拽式的使用方式，非常适合研究员使用，能够涵盖大部分分析研究的场景，如图 2-19 所示。但是，由于不能提供经典统计和机器学习算法支持，因此它不能代替统计和数据挖掘软件。另外，就实际处理速度而言，当面对较大数据（实例超过 3000 万记录）时，并没有官方介绍得那么迅速。

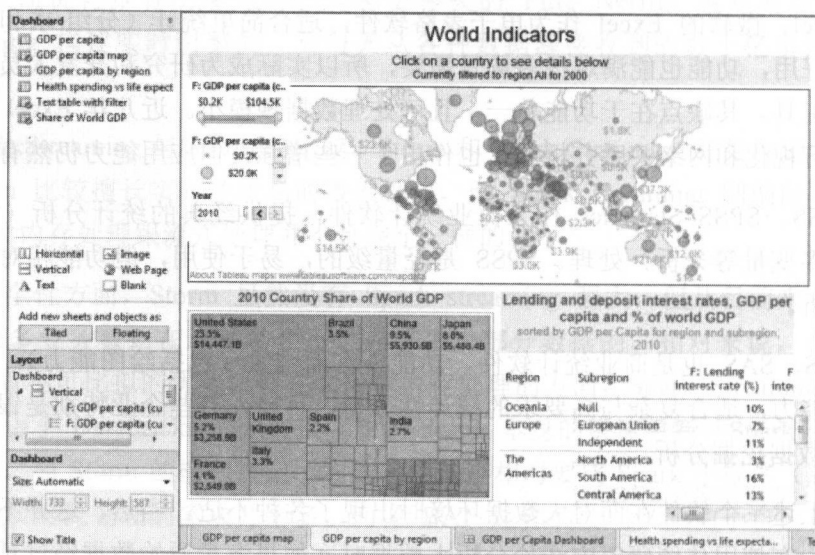


图 2-19 Tableau 的应用

关系分析工具 Gephi 是大数据环境下的一个新的分析热点（如信息传播图、社交关系网等），其计算的是点之间的关联关系。相关工具中，Gephi 是最适合数据研究人员的可视化轻量桌面型工具，如图 2-20 所示。Gephi 是免费软件，擅长解决图网络分析的很多需求，其插件众多，功能强且易用。我们经常看到的各种社交关系、传播谱图，都是基于

力导向图 (Force Directed Graph) 功能生成的。但由于该软件由 Java 编写, 限制了处理性能 (处理超过 10 万节点/边时常陷入假死), 所以, 在分析百万级节点 (如微博热点传播路径) 关系时, 需先做平滑和剪枝处理。而要处理更大规模 (如亿级以上) 的关系网络 (如社交网络关系) 数据, 则需要专门的图关系数据库 (如 GraphLab/GraphX) 来支撑。

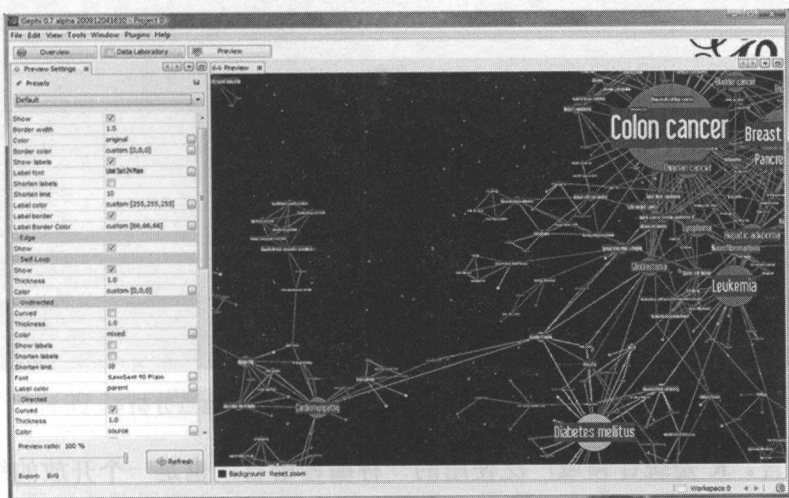


图 2-20 Gephi 应用场景

当前, 很多软件 (包括 Tableau) 都提供时空数据的可视化分析功能, 但大都只适合较小规模 (万级) 的可视化展示分析, 很少支持不同粒度的快速聚合探索。

如果要分析千万级以上的时空数据, 如新浪微博上亿用户发文的时间与地理分布 (从省到街道的多级粒度探索) 时, 可以使用 NanoCubes。该开源软件可在普通 PC 上提供对亿级时空数据的快速展示和多级实时钻取探索分析。图 2-21 是对芝加哥犯罪时间地点的分析, 网站有更多的实时分析的演示例子。

基于自然语言处理 (NLP) 的文本分析, 在非结构化内容 (如互联网/社交媒体/电商评论) 大数据的分析方面有重要用途, 其应用处理涉及分词、特征抽取、情感分析、多主题模型等众多内容。由于实现难度与领域差异, 当前市面上只有一些开源函数包或者云 API (如 BosonNLP) 提供一些基础处理功能, 尚未看到适合商业研究分析中文文本的集成化工具软件。在这种情况下, 各商业公司 (如 HCR) 主要依靠内部技术实力自主研发适合业务所需的分析功能。

4. 基于编程的大数据分析——R、Java、Python

前面介绍的各种大数据分析工具, 可应对的数据都在亿级以下, 也以结构化数据为

主。当实际面临亿级以上、半实时性处理、非标准化复杂需求时，通常就需要借助编程，并结合 Hadoop/Spark 等分布式计算框架来完成相关的分析。下面介绍几个当前适合大数据处理的编程语言。

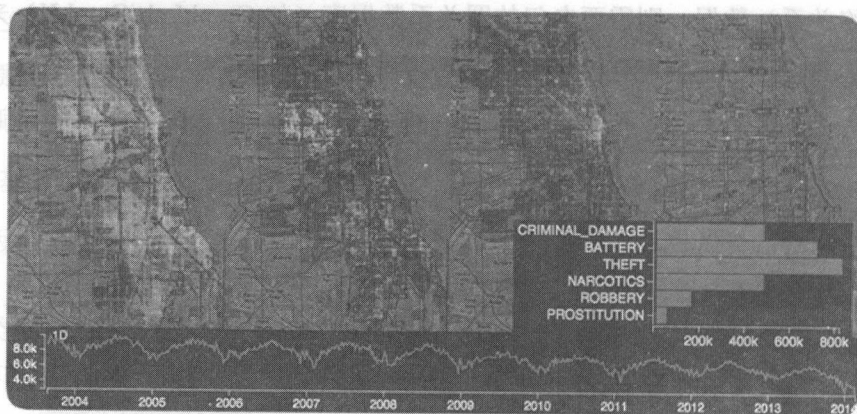


图 2-21 NanoCubes 用于芝加哥犯罪时间地点分析

(1) R 语言。R 是统计领域广泛使用的一种编程语言，也是一个开放的统计分析和图形显示的程序设计环境。与其说 R 是一种统计软件，还不如说 R 是一种数学计算的环境，因为 R 不仅提供了若干统计程序和工具，使用者只需指定数据库和若干参数便可进行一个统计分析，还提供了大量的数学计算、统计计算的函数，包括线性和非线性建模，经典的统计测试，时间序列分析、分类、收集，提供数组运算工具（其向量、矩阵运算方面功能尤其强大）。这些特点使得用户能灵活机动的进行数据分析，甚至创造出符合需要的新的统计计算方法。

R 具有丰富的统计分析功能库，以及可视化绘图函数可以直接调用，通过 Hadoop-R 更可支持处理百亿级别的数据。相比 SAS，其计算能力更强，可解决更复杂、更大数据规模的问题。R 是一个免费的自由软件，有 UNIX、Linux、MacOS 和 Windows 版本。R 的安装程序中只包含 8 个基础模块，其他模块可以通过 CRAN 获得。

R 语言可以直接访问 Hadoop 数据，可以实现全表、全字段立体式的数据挖掘。企业利用 R 语言的机器学习算法，可以开发深度学习应用，实现快速风险分析、市场营销、差别化服务和精细化客户管理。

(2) Python 语言。Python 是一种面向对象的解释型计算机程序设计语言，Python 语言在简洁性、易读性以及可扩展性等方面均非常好，因此众多开源的科学计算软件包都提供了 Python 的调用接口，例如著名的计算机视觉库 OpenCV、三维可视化库 VTK、医学

图像处理库 ITK 等, 而 Python 专用的科学计算扩展库就更多了, 例如十分经典的科学计算扩展库 NumPy、SciPy 和 matplotlib 等, 它们分别为 Python 提供了快速数组处理、数值运算以及绘图等功能。总体上, Python 语言及其众多的扩展库所构成的开发环境十分适合工程技术、科研人员处理实验数据、制作图表, 甚至开发科学计算应用程序。Python 代替 R 的势头越来越明显。

(3) Java 语言。Java 是通用性编程语言, 能力最全面, 拥有最多的开源大数据处理资源(统计、机器学习、NLP 等), 并得到所有分布式计算框架(Hadoop/Spark)的支持。

(4) Scala 语言。Scala 是一种类似 Java 的编程语言, 设计初衷是实现可伸缩的、集成面向对象编程和函数式编程的各种特性。Scala 融合了静态类型系统、面向对象、函数式编程等多种语言特性。Scala 是面向对象的编程语言, 所有的变量和方法都封装在对象中。Scala 同时又是函数式编程语言, 函数可以独立存在, 可以定义一个函数作为另一个函数的返回值, 也可以接收函数作为函数的参数。这给组合函数带来了很大的便利。Scala 无缝集成已有的 Java 类库, 用户可以非常自然地使用已经存在的非常庞大且稳定的 Java 类库。

以上介绍的面向大数据研究的不同工具、软件、语言, 各有自己的特点和适用场景。这些工具能够极大增强研究员在大数据环境下的分析能力, 但更重要的是数据分析人员要发挥自身对业务的深入理解, 从数据结果中洞察发现有深度的结果。

2.5.2 机器学习

复杂的大数据分析主要依靠机器学习。机器学习包括监督学习、非监督学习、强化学习等, 而监督学习又包括分类学习、回归学习、排序学习、匹配学习等。

1. 机器学习基础

机器学习(Machine Learning, ML)是一门多领域交叉学科, 涉及概率论、统计学、逼近论、凸分析、算法复杂度理论等多门学科, 它专门研究计算机怎样模拟或实现人类的学习行为, 以获取新的知识或技能, 重新组织已有的知识使之不断改善自身的结构和性能。机器学习是人工智能的核心, 是使计算机具有智能的根本途径, 其应用遍及人工智能的各个领域。机器学习主要使用归纳、综合, 而不是演绎。

综合考虑各种学习方法出现的历史渊源、知识表示、推理策略、结果评估的相似性、研究人员交流的相对集中性以及应用领域等诸因素, 可以将机器学习方法区分为以下六类。

(1) 经验性归纳学习: 经验性归纳学习(Empirical Inductive Learning)采用一些数据

密集的经验方法（如版本空间法、ID3 法、定律发现方法）对例子进行归纳学习，其例子和学习结果一般都采用属性、谓词、关系等符号表示，它相当于基于学习策略分类中的归纳学习，但扣除连接学习、遗传算法、加强学习的一部分。

（2）分析学习：分析学习（Analytic Learning）方法是从一个或少数几个实例出发，运用领域知识进行分析。其主要特征为：推理策略主要是演绎，而非归纳；使用过去的问题求解经验（实例）指导新的问题求解，或产生能更有效地运用领域知识的搜索控制规则。分析学习的目标是改善系统的性能，而不是新的概念描述，分析学习包括应用解释学习、演绎学习、多级结构组块及宏操作学习等技术。

（3）类比学习：它相当于基于学习策略分类中的类比学习。在这一类型的学习中比较引人注目的研究是通过与过去经历的具体事例作类比来学习，称为基于范例的学习（Case-Based Learning），或简称范例学习。

（4）遗传算法：遗传算法（Genetic Algorithm）模拟生物繁殖的突变、交换和达尔文的自然选择（在每一生态环境中适者生存）。它把问题可能的解编码为一个向量，称为个体，向量的每一个元素称为基因，并利用目标函数（相应于自然选择标准）对群体（个体的集合）中的每一个个体进行评价，根据评价值（适应度）对个体进行选择、交换、变异等遗传操作，从而得到新的群体。遗传算法适用于非常复杂和困难的环境，例如，带有大量噪声和无关数据、事物不断更新、问题目标不能明显和精确地定义，以及通过很长的执行过程才能确定当前行为的价值等。同神经网络一样，遗传算法的研究已经发展为人工智能的一个独立分支，其代表人物为霍勒德（Holland）。

（5）连接学习：典型的连接模型实现为人工神经网络，其由称为神经元的一些简单计算单元以及单元间的加权连接组成。近年来，基于卷积神经网络（CNN）、循环神经网络（RNN）的深度学习受到广泛关注。

（6）增强学习：增强学习（Reinforcement Learning）的特点是通过与环境的试探性（Trial And Error）交互来确定和优化动作的选择，以实现所谓的序列决策任务。在这种任务中，学习机制通过选择并执行动作，导致系统状态的变化，并有可能得到某种强化信号（立即回报），从而实现与环境的交互。强化信号就是对系统行为的一种标量化的奖惩。系统学习的目标是寻找一个合适的动作选择策略，即在任一给定的状态下选择哪种动作的方法，使产生的动作序列可获得某种最优的结果（如累计立即回报最大）。

在综合分类中，经验性归纳学习、遗传算法、连接学习和增强学习均属于归纳学习，其中经验性归纳学习采用符号表示方式，而遗传算法、连接学习和加强学习则采用亚符号表示方式；分析学习属于演绎学习。

实际上, 类比策略可看成归纳和演绎策略的综合, 因而最基本的学习策略只有归纳和演绎。

从学习内容的角度看, 采用归纳策略的学习由于是对输入进行归纳, 所学习的知识显然超过原有系统知识库所能蕴涵的范围, 所学结果改变了系统的知识演绎闭包, 因而这种类型的学习又可称为知识级学习; 而采用演绎策略的学习, 尽管所学的知识能提高系统的效率, 但仍能被原有系统的知识库所蕴涵, 即所学的知识未能改变系统的演绎闭包, 因而这种类型的学习又被称为符号级学习。

分类是最常见的机器学习应用问题, 如垃圾邮件过滤、人脸检测、用户画像、文本情感分析、网页归类等, 本质上都是分类问题。分类学习也是机器学习领域研究最彻底、使用最广泛的一个分支。最近, Fernández-Delgado 等人在 JMLR (Journal of Machine Learning Research, 机器学习顶级期刊) 杂志发表了一篇有趣的论文: 他们让 179 种不同的分类学习方法 (分类学习算法) 在 UCI 121 个数据集上进行了“大比武”(UCI 是机器学习公用数据集, 每个数据集的规模都不大), 结果发现 Random Forest (随机森林) 和 SVM (支持向量机) 名列第一、第二名, 但两者差异不大; 在 84.3% 的数据上, Random Forest 压倒其他 90% 的方法; 也就是说, 在大多数情况下, 只用 Random Forest 或 SVM 事情就搞定了。

关于机器学习算法的选择, 人们会问, 大数据分析到底需要多少种机器学习的方法呢? 围绕着这个问题, 我们看一下机器学习领域多年得出的一些经验规律。

大数据分析性能的好坏, 也就是说机器学习预测的准确率, 与使用的学习算法、问题的性质、数据集的特性, 包括数据规模、数据特征等都有关系。

一般而言, Ensemble 方法包括 Random Forest 和 AdaBoost、SVM、Logistic Regression 分类准确率最高。

没有一种方法可以包打天下。Random Forest、SVM 等方法一般性能最好, 但在任何条件下性能都最好。

不同的方法, 当数据规模较小的时候, 性能往往有较大差异, 但当数据规模增大时, 性能都会逐渐提升且差异逐渐减小。也就是说, 在大数据条件下, 什么方法都能工作得不错。

对于简单问题, Random Forest、SVM 等方法基本可行。但是, 对于复杂问题, 如语音识别、图像识别, 最近流行的深度学习方法往往效果更好, 深度学习本质是复杂模型学习。

在实际应用中，要提高分类的准确率，选择特征比选择算法更重要。好的特征会带来更好的分类结果，而好的特征的提取需要对问题的深入理解。

建立大数据分析平台时，选择实现若干种有代表性的方法即可。当然，不仅要考虑预测的准确率，还有考虑学习效率、开发成本、模型可读性等其他因素。大数据分析平台固然重要，同时需要有一批能够深入理解应用问题，自如使用分析工具的工程师和分析人员。

总之，只有善工利器，大数据分析才能真正发挥威力。

2. Mahout 算法库

机器学习算法库，包括朴素贝叶斯分类器、KNN 分类器、SVM、决策树、Boosting、梯度下降 Boosted 树、随机森林、EM 算法、神经网络。

Apache Mahout 是 Apache Software Foundation (ASF) 旗下的一个开源项目，提供一些可扩展的机器学习领域经典算法的实现，旨在帮助开发人员更加方便、快捷地创建智能应用程序，并且在 Mahout 的最近版本中还加入了对 Apache Hadoop 的支持，使这些算法可以更高效地运行在云计算环境中。Mahout 提供机器学习的各种算法，主要包括聚类、分类和协同过滤等算法，表 2-6 是 Mahout 算法库的算法列表。

表 2-6 Mahout 算法库的算法列表

算 法 类	算 法 名	中 文 名
分类算法	Logistic Regression	逻辑回归
	Bayesian	贝叶斯
	Support Vector Machines	支持向量机
	Perceptron and Winnow	感知器算法
	Neural Network	神经网络
	Random Forests	随机森林
	Restricted Boltzmann Machines	有限波尔兹曼机
聚类算法	Canopy Clustering	Canopy 聚类
	K-Means Clustering	K 均值算法
	Fuzzy K-Means	模糊 K 均值
	Expectation Maximization	EM 聚类（期望最大化聚类）
	Mean Shift Clustering	均值漂移聚类
	Hierarchical Clustering	层次聚类
	Dirichlet Process Clustering	狄里克雷过程聚类
	Latent Dirichlet Allocation	LDA 聚类
	Spectral Clustering Minhash Clustering Top Down Clustering	谱聚类
关联规则挖掘	Parallel FP Growth Algorithm	并行 FP Growth 算法

续表

算 法 类	算 法 名	中 文 名
回归	Locally Weighted Linear Regression	局部加权线性回归
降维/维约简	Stochastic Singular Value Decomposition	奇异值分解
	Principal Components Analysis	主成分分析
	Independent Component Analysis	独立成分分析
	Gaussian Discriminative Analysis	高斯判别分析
进化算法	并行化了 Watchmaker 框架	
推荐/协同过滤	Non-distributed recommenders	Taste (UserCF、ItemCF、SlopeOne)
	Distributed Recommenders	ItemCF
向量相似度计算	RowSimilarityJob	计算列间相似度
	VectorDistanceJob	计算向量间距离
非 Map-Reduce 算法	Hidden Markov Models	隐马尔科夫模型
集合方法扩展	Collocations	扩展了 Java 的 Collections 类

Mahout 最大的优点就是基于 Hadoop 实现，它把很多以前运行于单机上的算法，转化为 MapReduce 模式，这样就大大提升了算法可处理的数据量和处理性能。通过和 Apache Hadoop 分布式框架相结合，Mahout 可以有效地使用分布式系统来实现高性能计算。

3. Spark MLlib 库

Spark MLlib (Machine Learning lib) 是基于 Spark 的一个机器学习库，它提供了各种各样的算法，这些算法用来在集群上针对分类、回归、聚类、协同过滤等。其中一些算法也可以应用到流数据上，例如使用普通最小二乘法或者 K 均值聚类（还有更多）来计算线性回归。值得注意的是，Apache Mahout 已经脱离 MapReduce，转而加入 Spark MLlib。

Spark 之所以在机器学习方面具有得天独厚的优势，有以下两点原因。

(1) 机器学习算法一般都有很多个步骤迭代计算的过程，机器学习的计算需要在多次迭代后获得足够小的误差或者足够收敛才会停止，迭代时如果使用 Hadoop 的 MapReduce 计算框架，每次计算都要读/写磁盘，以及任务的启动等工作，这会导致非常大的 I/O 和 CPU 消耗。而 Spark 基于内存的计算模型天生就擅长迭代计算，多个步骤计算直接在内存中完成，只有在必要时才会操作磁盘和网络，所以说 Spark 正是机器学习的理想的平台。

(2) 从通信的角度讲，如果使用 Hadoop 的 MapReduce 计算框架，JobTracker 和 TaskTracker 之间由于是通过 heartbeat 的方式来进行的通信和传递数据，会导致非常慢的执行速度，而 Spark 具有出色而高效的通信系统，通信效率很高。

MLlib 是 Spark 对常用的机器学习算法的实现库，同时包括相关的测试和数据生成器。Spark 的设计初衷就是为了支持一些迭代的 Job，这正好符合很多机器学习算法的特点。

MLlib 基于 RDD，可以与 Spark SQL、GraphX、Spark Streaming 无缝集成。

2.6 本章小结

本章介绍了大数据系统涉及的关键技术，主要包括数据采集与生成、数据分布式存储、分布式计算框架、数据分析与挖掘等方面的技术和工具。

(1) 数据来源：例如，银行大数据系统的数据来源包括银行内部业务系统产生的结构化数据和非结构化数据，以及银行外部的海量数据。

(2) 数据采集与生成：业务系统的结构化数据到大数据平台（以 Hadoop 为核心）的导入工具（Sqoop），日志数据的采集与导入工具 Flume，数据分发工具 Kafka。

(3) 数据存储基本概念与海量数据分布式存储技术：介绍了列存储，K-V 存储与 NoSQL 数据库等基本概念，介绍了分布式文件存储 HDFS 和分布式内存文件存储 Tachyon，以及数据库 HBase 和数据仓库 Hive。

(4) 介绍了分布式计算框架：主要介绍了离线计算框架 MapReduce 和 YARN，对其中的架构和原理进行了深入介绍，并详细介绍了 Spark 和 Flink。同时，本章还详细介绍了实时流计算平台 Storm 和 Spark Streaming，对其架构、实现机制与适用环境等进行了深入的分析。

(5) 数据挖掘平台与工具：详细介绍了数据挖掘与分析工具，对面向大数据的数据挖掘常用工具进行了分类。

(6) 本章最后简要介绍了机器学习及 Mahout、Spark MLlib 算法库。

本章主要从各组件的构成、原理、实现机制和适用条件等方面进行了阐述和分析，以帮助读者从整体上对大数据架构中的各个组件有基本的了解。后续各章将帮助读者深入了解主要组件的安装、配置和应用等。

第二篇

Hadoop 大数据平台搭建与基本应用

Hadoop 大数据系统主要在 Linux 平台上运行，因此，本章介绍 Linux 操作系统的基本知识和操作方法。重点介绍 Linux 集群的安装和搭建、Java 开发包（JDK）的安装，以及集群的基本配置方法，为后续实践搭建平台。

3.1 Linux 操作系统

3.1.1 概述

Linux 的出现，最早在 20 世纪 90 年代初，一位名叫 Linus Torvalds 的计算机业余爱好者开发了该系统，当时他是芬兰赫尔辛基大学的学生。他的目的是想设计一个代替 Minix（是由一位名叫 Andrew Tanenbaum 的计算机教授编写的一个操作系统示教程序）的操作系统，这个操作系统可用于 386、486 或奔腾处理器的个人计算机上，并且具有 UNIX 操作系统的全部功能，因而开始了 Linux 雏形的设计。

Linux 以它的高效性和灵活性著称，它能够在 PC 计算机上实现全部的 UNIX 特性，具有多任务、多用户的能力。Linux 是在 GNU 公共许可权限下免费获得的，是一个符合 POSIX 标准的操作系统。Linux 操作系统软件包不仅包括完整的 Linux 操作系统，而且包括了文本编辑器、高级语言编译器等应用软件。它还包括带有多个窗口管理器的 X-Windows 图形用户界面，如同我们使用 Windows NT 一样，允许我们使用窗口、图标和菜单对系统进行操作。

Linux 之所以受到广大计算机爱好者的喜爱，主要原因有两个，一是它属于自由软件，用户不用支付任何费用就可以获得它和它的源代码，并且可以根据自己的需要对它进

行必要的修改，无偿使用，无约束地继续传播；另一个原因是，它具有 UNIX 的全部功能，任何使用 UNIX 操作系统或想要学习 UNIX 操作系统的人都可以从 Linux 中获益。

简单地说，Linux 是一套免费使用和自由传播的类 UNIX 操作系统，是一个基于 POSIX 与 UNIX 的多用户、多任务、支持多线程和多 CPU 的操作系统。它能运行主要的 UNIX 工具软件、应用程序和网络协议，它支持 32 位和 64 位硬件。Linux 继承了 UNIX 以网络为核心的设计思想，是一个性能稳定的多用户网络操作系统，它主要用于基于 Intel x86 系列 CPU 的计算机上。这个系统是由全世界各地的成千上万的程序员设计和实现的，其目的是建立不受任何商品化软件的版权制约的、任何用户都能自由使用的 UNIX 兼容产品。

3.1.2 特点

Linux 操作系统在短时间内得到了非常迅猛的发展，这与 Linux 具有的良好特性是分不开的。Linux 包含了 UNIX 的全部功能和特性。归纳起来，Linux 具有以下主要特性。

(1) 开放性：开放性是指系统遵循世界标准规范，特别是遵循开放系统互连（OSI）国际标准。凡遵循国际标准所开发的硬件和软件，都能彼此兼容，可方便地实现互连。

(2) 多用户：多用户是指系统资源可以被不同用户各自拥有使用，即每个用户对自己的资源（如文件、设备）有特定的权限，互不影响。Linux 和 UNIX 都具有多用户的特性。

(3) 多任务：多任务是现代计算机的最主要的一个特点，它是指计算机同时执行多个程序，而且各个程序的运行互相独立。Linux 系统调度每一个进程平等地访问微处理器。由于 CPU 的处理速度非常快，其结果是启动的应用程序看起来好像在并行运行。事实上，从处理器执行一个应用程序中的一组指令到 Linux 调度微处理器再次运行这个程序之间只有很短的时间延迟，用户是感觉不出来的。

(4) 良好的用户界面：Linux 向用户提供了两种界面——用户界面和系统调用。Linux 的传统用户界面是基于文本的命令行界面，即 Shell，它既可以联机使用，又可存在文件上脱机使用。Shell 有很强的程序设计能力，用户可方便地用它编制程序，从而为扩充系统功能提供了更高级的手段。可编程 Shell 是指将多条命令组合在一起，形成一个 Shell 程序，这个程序可以单独运行，也可以与其他程序同时运行。

系统调用给用户编程时使用的界面，用户可以在编程时直接使用系统提供的系统调用命令，系统通过这个界面为用户程序提供低级、高效率的服务。

Linux 还为用户提供了图形用户界面,它利用鼠标、菜单、窗口、滚动条等设施,给用户呈现一个直观、易操作、交互性强的友好的图形化界面。

(5) 设备独立性:设备独立性是指操作系统把所有外部设备统一当成文件来看待,只要安装它们的驱动程序,任何用户都可以像使用文件一样操纵、使用这些设备,而不必知道它们的具体存在形式。

具有设备独立性的操作系统,通过把每一个外围设备看作一个独立文件来简化增加新设备的工作。当需要增加新设备时,系统管理员就在内核中增加必要的连接。这种连接(也称作设备驱动程序)保证每次调用设备提供服务时,内核以相同的方式来处理它们。当新的及更好的外设被开发并交付给用户时,操作允许在这些设备连接到内核后,就能不受限制地立即访问它们。设备独立性的关键在于内核的适应能力,其他操作系统只允许一定数量或一定种类的外部设备连接,而设备独立性的操作系统能够容纳任意种类及任意数量的设备,因为每一个设备都是通过其与内核的专用连接独立进行访问的。

Linux 是具有设备独立性的操作系统,它的内核具有高度适应能力,随着更多的程序员加入 Linux 编程,会有更多硬件设备加入到各种 Linux 内核和发行版本中。另外,由于用户可以免费得到 Linux 的内核源代码,因此,用户可以修改内核源代码,以便适应新增加的外部设备。

(6) 丰富的网络功能,完善的内置网络是 Linux 的一大特点, Linux 在通信和网络功能方面优于其他操作系统。其他操作系统不包含如此紧密地和内核结合在一起的连接网络的能力,也没有内置这些联网特性的灵活性。而 Linux 为用户提供了完善的、强大的网络功能。

支持 Internet 是其网络功能之一。Linux 免费提供了大量支持 Internet 的软件,Internet 是在 UNIX 领域中建立并繁荣起来的,在这方面使用 Linux 是相当方便的,用户能用 Linux 与世界上的其他人通过 Internet 网络进行通信。文件传输是其网络功能之二,用户能通过一些 Linux 命令完成内部信息或文件的传输。远程访问是其网络功能之三, Linux 不仅允许进行文件和程序的传输,它还为系统管理员和技术人员提供了访问其他系统的窗口,通过这种远程访问的功能,一位技术人员能够有效地为多个系统服务,即使那些系统位于相距很远的地方。

(7) 可靠的系统安全:Linux 采取了许多安全技术措施,包括对读、写进行权限控制、带保护的子系统、审计跟踪、核心授权等,这为网络多用户环境中的用户提供了必要的安全保障。

(8) 良好的可移植性：可移植性是指将操作系统从一个平台转移到另一个平台时仍然能按其自身的方式运行的能力。Linux 是一种可移植的操作系统，能够在从微型计算机到大型计算机的任何环境中中和任何平台上运行。可移植性为运行 Linux 的不同计算机平台与其他任何机器进行准确而有效的通信提供了手段，不需要另外增加特殊的和昂贵的通信接口。

3.1.3 Linux 的组成

操作系统是一台计算机必不可少的系统软件，是整个计算机系统的灵魂。一个操作系统是一个复杂的计算机程序集，它提供操作过程的协议或行为准则。没有操作系统，计算机就无法工作，就不能解释和执行用户输入的命令或运行简单的程序。

Linux 包括四个组成部分：内核、Shell、文件系统和实用工具。

1. 内核

Linux 是一个整体化内核 (Monolithic Kernel) 系统。“内核”指的是提供硬件抽象层、磁盘及文件系统控制、多任务等功能的系统软件；在 Linux 的术语中被称为“内核”，也可以称为“核心”。

一个内核不是一套完整的操作系统。Linux 内核的主要模块 (或组件) 分为以下几个部分：存储管理、CPU 和进程管理、文件系统、设备管理和驱动、网络通信，以及系统的初始化 (引导)、系统调用等。一套基于 Linux 内核的完整操作系统叫做 Linux 操作系统，或者 GNU/Linux。Linux 内的设备驱动程序可以方便地以模块化 (Modularize) 的形式设置，并在系统运行期间可直接装载或卸载。

2. Shell

Shell 是系统的用户界面，提供了用户与内核进行交互操作的一种接口，它接收用户输入的命令并把它送入内核去执行。实际上 Shell 是一个命令解释器，它解释由用户输入的命令并且把它们送到内核。不仅如此，Shell 有自己的编程语言用于对命令的编辑，它允许用户编写由 Shell 命令组成的程序。Shell 编程语言具有普通编程语言的很多特点，比如它也有循环结构和分支控制结构等，用这种编程语言编写的 Shell 程序与其他应用程序具有同样的效果。

Linux 提供了像 Microsoft Windows 那样的可视的命令输入界面，称为 X Window 的图形用户界面 (GUI)。它提供了很多窗口管理器，其操作就像 Windows 一样，有窗口、图标和菜单，所有的管理都是通过鼠标控制的。现在比较流行的窗口管理器是 KDE 和 GNOME。

每个 Linux 系统的用户可以拥有他自己的用户界面或 Shell，用以满足他们自己专门的 Shell 需要。

同 Linux 本身一样，Shell 也有多种不同的版本。目前主要有下列版本的 Shell：Bourne Shell，是贝尔实验室开发的；BASH，是 GNU 的 Bourne Again Shell，是 GNU 操作系统上默认的 Shell；Korn Shell，这是 Bourne Shell 的发展，在大部分内容上与 Bourne Shell 兼容；C Shell，是 Sun 公司 Shell 的 BSD 版本。

3. 文件系统

文件系统是操作系统的重要组成部分，主要负责管理磁盘文件的输入输出。

文件通过目录方式进行组织，目录结构是文件存放在磁盘等存储设备上的组织方式，目录提供了管理文件的一个方便而有效的途径。用户可以设置目录和文件的权限，以便允许或拒绝其他人对其进行访问；设置文件的共享程度，能够从一个目录切换到另一个目录。

Linux 目录采用多级树形结构，图 3-1 表示了这种树形结构。用户可以浏览整个系统，进入任何一个已授权进入的目录，访问那里的文件。

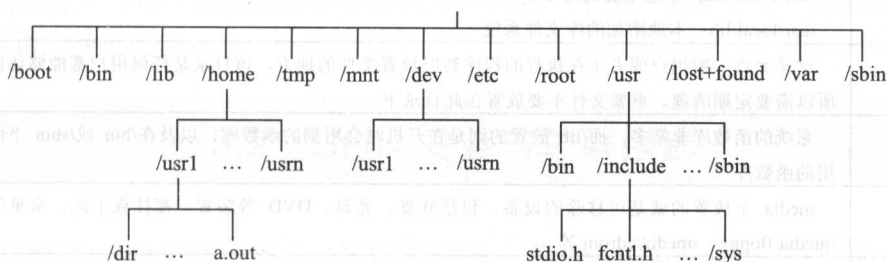


图 3-1 Linux 的树形目录等级结构

表 3-1 给出了 Linux 系统目录的内容与用途。

表 3-1 Linux 系统的目录与用途

目 录	存放的文件与内容
/bin	系统有很多放置执行文件的目录，但/bin 比较特殊。/bin 放置的是在单用户模式下也能够被操作的指令。/bin 下的指令可以被 root 和一般用户所使用，例如 cat、chmod、chown、date、mv、mkdir、cp、bash 等常用的命令
/boot	主要存放开机使用到的文件，包括 Linux 核心文件、开机选择、开机所需设定文件等
/dev	Linux 系统上，任何装置与周边设备都以文件形式存放于这个目录当中，只要通过存取这个目录下的某个文件，就等于存取某个设备。 比较重要的文件有/dev/null、/dev/zero、/dev/tty、/dev/lp*、/dev/hd*、/dev/sd*等

续表

目 录	存放的文件与内容
/etc	<p>系统主要的配置文件几乎都放置在这个目录内，例如人员账号密码文件、各种服务的起始文件等。一般来说，这个目录下的各文件属性是可以让一般使用户读取的，但只有 root 用户有权修改。</p> <p>比较重要的档案有/etc/passwd、/etc/shadow、/etc/modprobe.conf、/etc/X11/、/etc/fstab、/etc/sysconfig/等。另外，其下重要的子目录还有/etc/init.d/，所有服务的预设启动 script 都放在这里的，例如要启动或者关闭 iptables，可以执行/etc/init.d/iptables start、/etc/init.d/iptables stop</p>
/home	<p>这是系统预设的用户家目录（Home Directory）。在新增一个一般用户账号时，预设的用户家目录都会建立在这里。家目录有两种代号，~代表当前使用者的家目录，而~guest 则代表用户名为 guest 的家目录</p>
/usr	<p>/usr 是 Linux 系统中最重要目录之一，涵盖了二进制文件、各种文档、各种头文件、库文件，以及诸多程序，如 ftp、telnet 等。/usr 目录较大，要用到的应用程序和文件几乎都在这个目录中，其中包含：</p> <p>/usr/x11R6：存放 x window 的目录。</p> <p>/usr/bin：众多的应用程序。</p> <p>/usr/sbin：超级用户的一些管理程序。</p> <p>/usr/doc：Linux 文档。</p> <p>/usr/include：Linux 下开发和编译应用程序所需要的头文件。</p> <p>/usr/lib：常用的动态链接库和软件包的配置文件。</p> <p>/usr/man：帮助文档。</p> <p>/usr/src：源代码，Linux 内核的源代码就放在/usr/src/linux 里。</p> <p>/usr/local/bin：本地增加的命令。</p> <p>/usr/local/lib：本地增加的库文件系统</p>
/tmp	<p>这是允许一般用户或者正在执行的程序暂时放置文件的地方，该目录是任何用户都能够访问的目录，所以需要定期清理。重要文件不要放置在此目录下</p>
/lib	<p>系统的函数库非常多，而/lib 放置的则是在开机时会用到的函数库，以及在/bin 或/sbin 下面的指令调用的函数库</p>
/media	<p>media 下放置的就是可移除的设备，包括软盘、光盘、DVD 等装置，都挂载于此。常见的文件名有/media/floppy、/media/cdrom 等</p>
/mnt	<p>如果想要临时挂载某些额外的设备，一般建议放置到这个目录中。以前这个目录的用途与/media 相同，但有了/media 之后，这个目录就用来临时挂载设备了</p>
/opt	<p>存放第三方软件的目录</p>
/root	<p>系统管理员（root）的家目录。之所以放在这里，是因为如果进入单用户模式而仅挂载根目录时，该用户就能够拥有 root 的家目录，所以 root 的家目录与根目录放置在同一个分区中</p>
/sbin	<p>Linux 有很多指令是用来设置系统环境的，这些指令只有 root 用户才能够使用，其他用户只能查询。放在/sbin 下面的命令是启动过程中所需要的，包括了开机、修复、还原系统所需要的指令</p>
/srv	<p>Srv 是一些网路服务启动之后，这些服务所需要使用的文件的目录。</p> <p>常见的服务有 WWW、FTP 等。例如，WWW 服务器需要的网页文件就可以放置在/srv/www/里面</p>
/lost+found	<p>该目录是使用标准的 ext2/ext3 文件系统格式才会产生的目录，用于在文件系统发生错误时，将一些遗留的文件片段存放到该目录下</p>
/var	<p>/var 目录主要用于存放经常变动的文件，包括缓存文件（Cache）、日志文件（Log File）及某些软件运行产生的文件等</p>

4. 常用工具

Linux 实用工具可分三类：编辑器（用于编辑文件）、过滤器（用于接收数据并过滤数据）、交互程序（允许用户发送信息或接收来自其他用户的信息）。

（1）Linux 的编辑器主要有 gedit、Ex、Vi 和 Emacs。Ed 和 Ex 是行编辑器，Vi 和 Emacs 是全屏幕编辑器。

（2）Linux 的过滤器（Filter）读取从用户文件或其他地方的输入，检查和处理数据，然后输出结果。从这个意义上说，它们过滤了经过它们的数据。Linux 有不同类型的过滤器，一些过滤器用行编辑命令输出一个被编辑的文件。另外一些过滤器是按模式寻找文件并以这种模式输出部分数据的。还有一些执行字处理操作，检测一个文件中的格式，输出一个格式化的文件。过滤器的输入可以是一个文件，也可以是用户从键盘键入的数据，还可以是另一个过滤器的输出。过滤器可以相互连接，因此，一个过滤器的输出可能是另一个过滤器的输入。在有些情况下，用户可以编写自己的过滤器程序。

（3）交互程序是用户与机器的信息接口。Linux 是一个多用户系统，它必须和所有用户保持联系。信息可以由系统上的不同用户发送或接收，信息的发送有两种方式，一种方式是与其他用户一对一地连接进行对话，另一种是一个用户对多个用户同时连接进行通信，即所谓广播式通信。

3.2 Linux 安装与集群搭建

本节介绍在 Windows 平台上，通过 VMware Workstation 虚拟机安装 Linux 集群。需要指出的是，在同一台计算机上安装虚拟 Linux 集群有很多优点，首先，其使用方法与真实集群基本一致；其次，操作人员不必到实际的计算节点检查结果，可以节省大量学习时间。实际中，将工作站和服务器安装在虚拟机环境，可使系统管理简化、缩减实际的机房面积，并减少对硬件的需求。

3.2.1 安装 VMware Workstation

VMware（中文名“威睿”）是一家总部设在美国加利福尼亚州帕罗奥多市（Palo Alto）的软件公司，是全球桌面到数据中心虚拟化解决方案的领导厂商。全球不同规模的客户依靠 VMware 来降低建设和运营成本、确保业务持续性、加强安全性并走向绿色。VMware 是云计算时代增长最快的上市软件公司之一。

VMware 也是该公司开发的一套 PC 虚拟化软件的总称。VMware Workstation 是 VMware 系列软件产品之一，用于在英特尔 x86 兼容计算机上创建虚拟机工作站，它允许用户同时创建和运行多个 x86 虚拟机。每个虚拟机实例可以运行自己的客户机操作系统，如 Windows、Linux、BSD 变生版本等。简而言之，VMware Workstation 允许一台真实的计算机在一个操作系统中同时安装并运行多个操作系统，并帮助用户在多个宿主计算机之间管理或移植 VMware 虚拟机。

下面开始安装。

首先，请读者从网络下载免费的 VMware Workstation，也可以在本书第 3 章软件资源文件夹中找到 vmware 目录，进入该目录，并根据读者计算机 Windows 操作系统的版本选择进入 Win7、Win8 或 Win10 子目录，我们这里以 win10 下的安装为例，如图 3-2 所示。

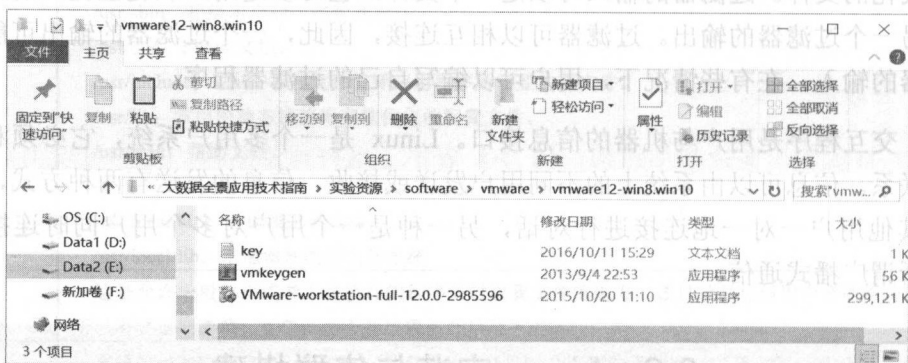


图 3-2 software\vmware 目录下的 VMware-workstation-full-12.0.0-2985596.exe

单击“VMware-workstation-full-12.0.0-2985596.exe”开始安装。

安装程序首先检测系统并解压文件，完成后出现“欢迎使用”界面，如图 3-3 (a) 所示。

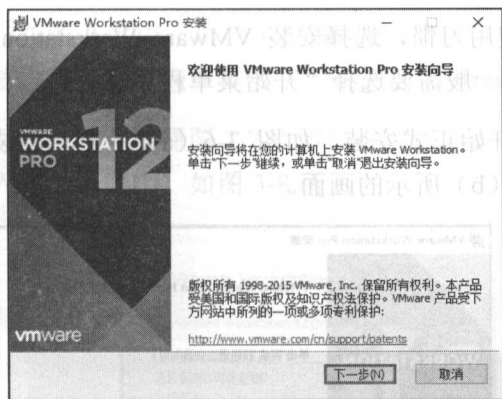
接着直接单击“下一步”按钮，显示“最终用户许可协议”画面，如图 3-3 (b) 所示，选择“我接受许可协议中的条款”，再单击“下一步”按钮即可。

接着出现图 3-4 所示的“自定义安装”界面。

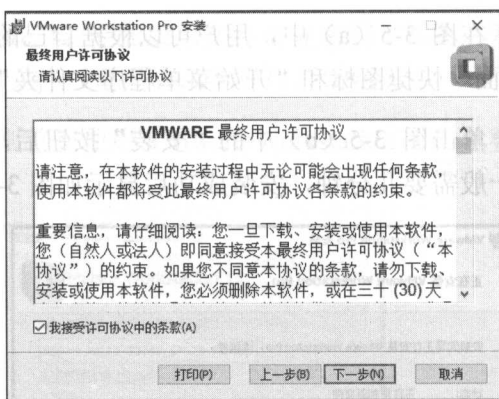
与 Win7 下的 VMware10 相比，VMware12 只提供“自定义安装”，不再显示“典型安装”或“自定义安装”选择界面。

在图 3-4 (a) 中，用户可以修改 VMware Workstation 的安装位置，建议直接选择缺省安装位置。而图 3-4 (b) 中的用户体验设置可以自行选择，通常不建议选择。

单击“下一步”按钮，进入如图 3-5 所示的界面。

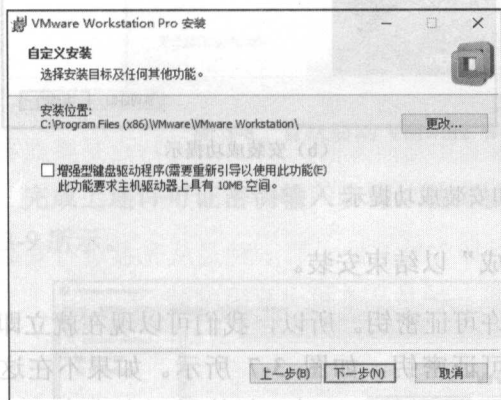


(a) 欢迎使用界面

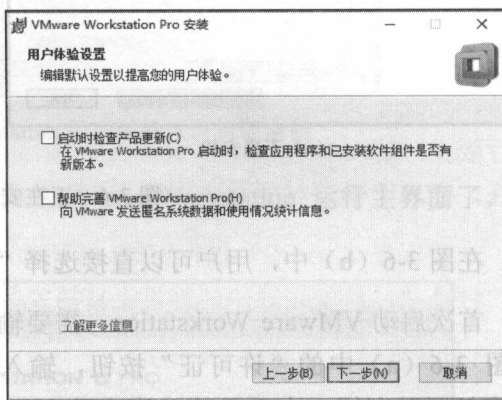


(b) 接受许可协议

图 3-3 VMware-workstation 安装向导

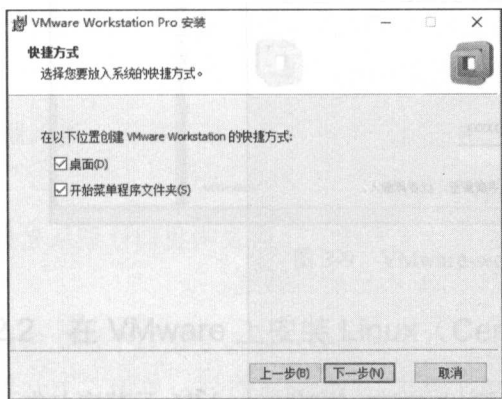


(a) 用于修改安装位置

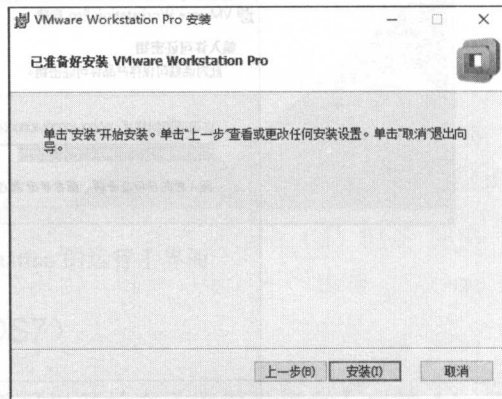


(b) 用于用户体验设置

图 3-4 自定义安装界面



(a) 用于选择“快捷方式”

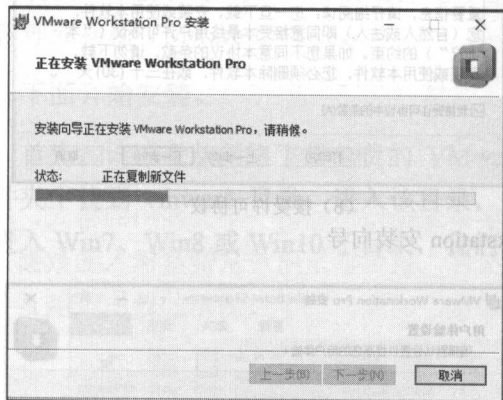


(b) 提示安装准备就绪

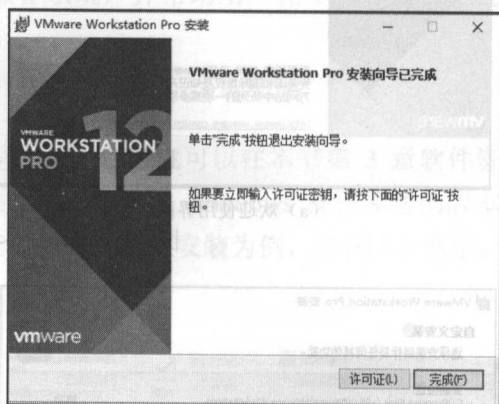
图 3-5 VMware Workstation Pro 安装界面

在图 3-5 (a) 中, 用户可以根据自己的使用习惯, 选择安装 VMware Workstation 的“桌面”快捷图标和“开始菜单程序文件夹”。一般要选择“开始菜单程序文件夹”。

单击图 3-5 (b) 中的“安装”按钮后, 开始正式安装, 如图 3-6 (a) 所示。安装过程一般需要几分钟, 安装成功后显示如图 3-6 (b) 所示的画面。



(a) 正确安装



(b) 安装成功提示

图 3-6 正在安装和安装成功提示

在图 3-6 (b) 中, 用户可以直接选择“完成”以结束安装。

首次启动 VMware Workstation, 需要输入许可证密钥。所以, 我们可以现在就立即单击图 3-6 (a) 中的“许可证”按钮, 输入许可证密钥, 如图 3-7 所示。如果不在这一步输入许可证密钥, 在后面第一次启动 VMware Workstation 时, 仍然需要完成许可证密钥的验证。

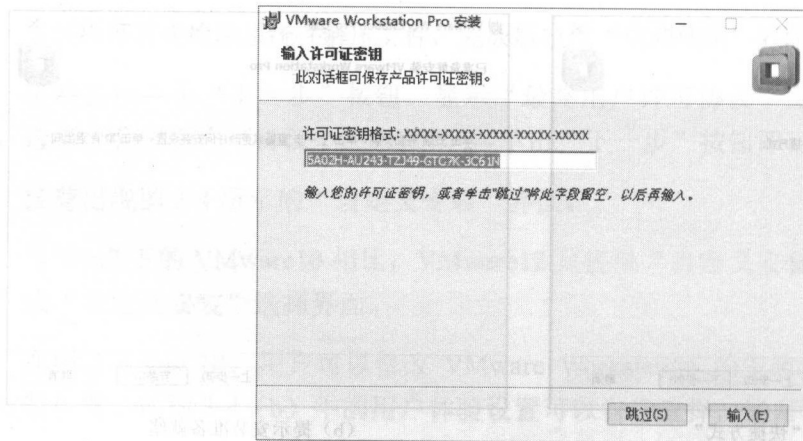


图 3-7 输入许可证密钥

在图 3-7 中,用户可以输入给定的 VMware 12 专业版永久许可证密钥(本书在第 3 章软件资源文件夹中提供了一个名为 key.txt 的文件,里面有永久许可证密钥)。

如果没有选择图 3-7 所示的许可证输入,则首次使用 VMware Workstation 时仍然要求输入许可证密钥,如图 3-8 所示。

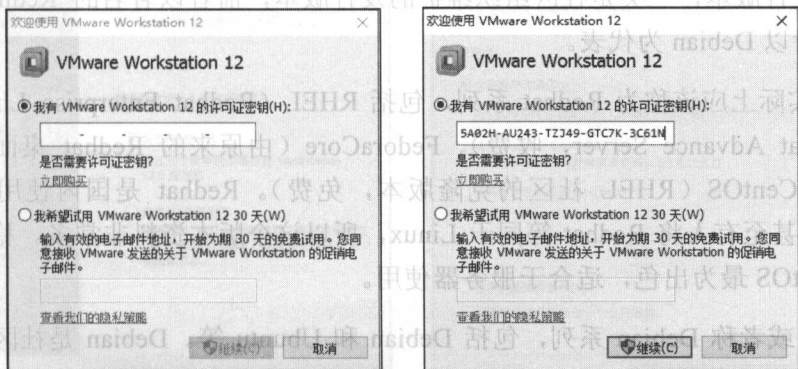


图 3-8 首次启动 VMware Workstation 的时候输入许可证密钥

完成上述许可证密钥输入步骤,就可以进入 VMware Workstation 运行主界面了,如图 3-9 所示。

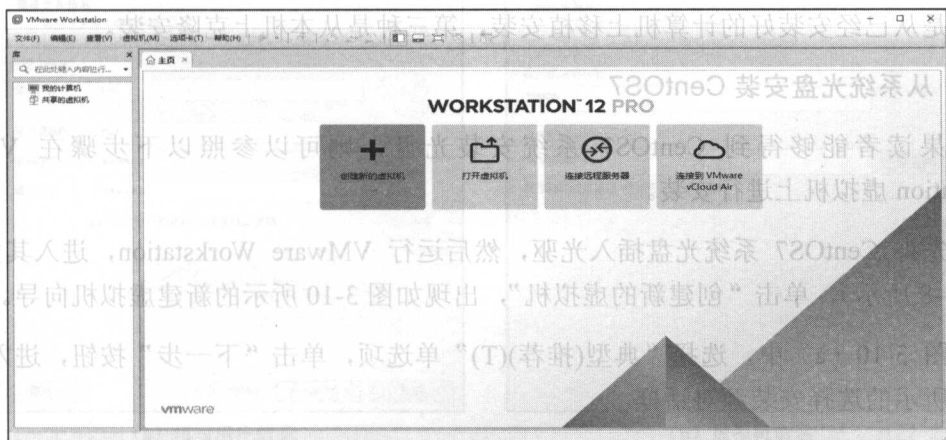


图 3-9 VMware-workstation 的运行主界面

3.2.2 在 VMware 上安装 Linux (CentOS7)

成功安装了 VMware Workstation 以后,就可以开始在该虚拟机平台上安装 Linux 操作系统了。

Linux 是开源操作系统，因此版本繁多。严格来讲，Linux 这个词本身只表示 Linux 内核，各种发行版为许多不同的目标而制作，包括对不同计算机结构的支持，具体区域或语言的本地化，实时应用，嵌入式系统，等等。已经有超过 300 个发行版被积极地开发，最普遍被使用的发行版有大约十来个。Linux 的发行版本可以大体分为两类，一类是商业公司维护的发行版本，一类是社区组织维护的发行版本，前者以著名的 Redhat (RHEL) 为代表，后者以 Debian 为代表。

Redhat 实际上应该称为 Redhat 系列，包括 RHEL (Redhat Enterprise Linux，也就是所谓的 Redhat Advance Server，收费)、FedoraCore (由原来的 Redhat 桌面版本发展而来，免费)、CentOS (RHEL 社区的克隆版本，免费)。Redhat 是国内使用人数最多的 Linux 版本，甚至有人将 Redhat 等同于 Linux，所以这个版本资料非常多。稳定性方面，RHEL 和 CentOS 最为出色，适合于服务器使用。

Debian，或者称 Debian 系列，包括 Debian 和 Ubuntu 等。Debian 是社区类 Linux 的典范，是迄今为止最遵循 GNU 规范的 Linux 系统。严格来说，Ubuntu 不能算一个独立的发行版本，Ubuntu 是基于 Debian 的 unstable 版本加强而来的。Ubuntu 的特点是界面非常友好，容易上手，对硬件的支持全面，是最适合做桌面系统的 Linux 发行版本。

本书选择 CentOS7 进行安装。我们介绍三种安装方法，第一种是从光盘进行安装，第二种是从已经安装好的计算机上移植安装，第三种是从本机上克隆安装。

1. 从系统光盘安装 CentOS7

如果读者能够得到 CentOS7 系统安装光盘，则可以参照以下步骤在 VMware Workstation 虚拟机上进行安装。

首先将 CentOS7 系统光盘插入光驱，然后运行 VMware Workstation，进入其主界面 (如图 3-8 所示)，单击“创建新的虚拟机”，出现如图 3-10 所示的新建虚拟机向导。

在图 3-10 (a) 中，选择“典型(推荐)(T)”单选项，单击“下一步”按钮，进入图 3-9 (b) 所示的选择安装源对话框。

在图 3-10 (b) 中，选择第一个选项“安装程序光盘(D)”，这表明我们计划从光盘安装 CentOS7，单击“下一步”按钮，出现图 3-11 (a) 所示的对话框。

在图 3-11 (a) 中，用户需要填写账号信息，可以根据自己的需要填写个性化信息，例如，全名：csu，用户名：csu，密码：csucsu，确认：csucsu。然后，单击“下一步”按钮，进入如图 3-11 (b) 所示的虚拟机命名与安装位置选择对话框。在图 3-11 (b)

中，我们修改了缺省的虚拟机名称，将虚拟机命名为 Master，安装位置也修改为“D:\Master”。实际上，用户可以根据自己的需要进行任意修改。单击“下一步”按钮，进入如图 3-12 所示的对话框。

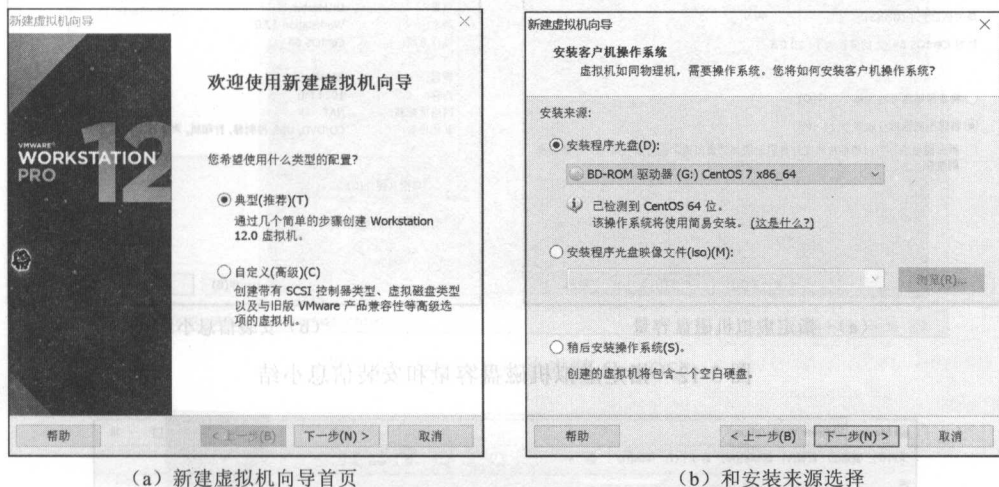


图 3-10 新建虚拟机向导首页和安装来源选择

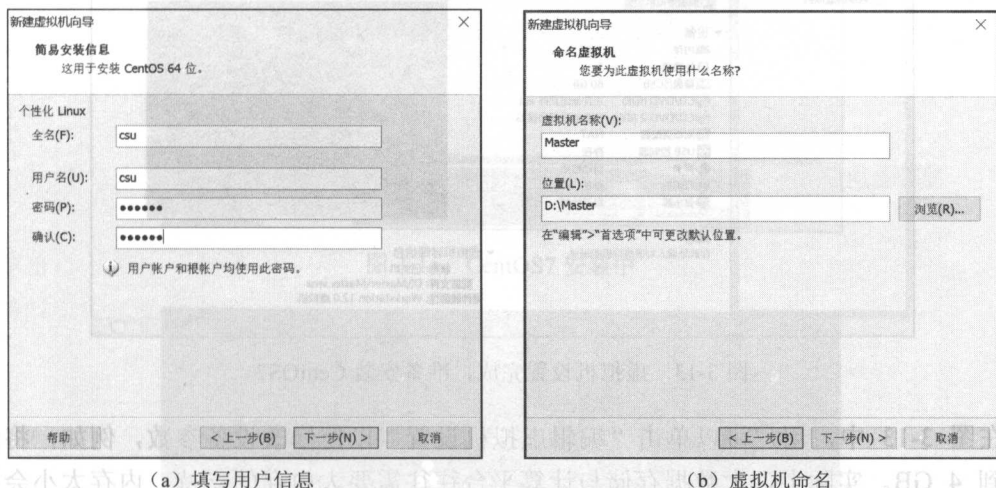
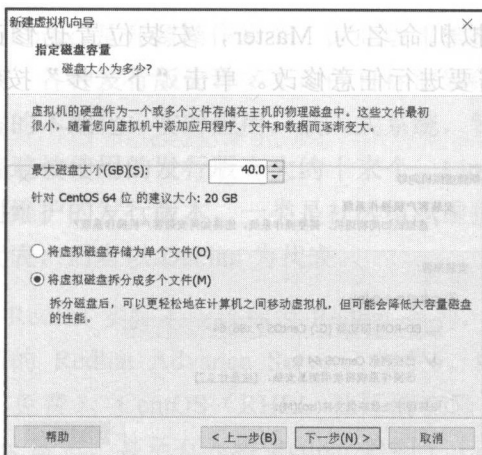
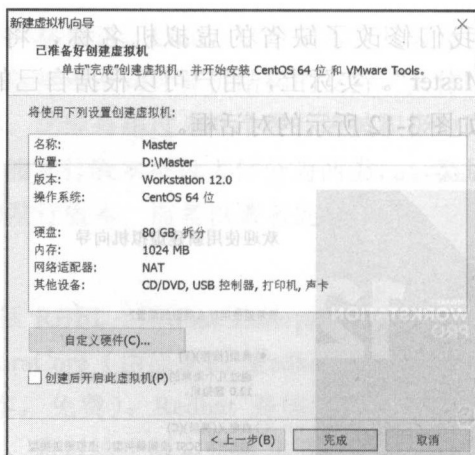


图 3-11 填写用户信息与虚拟机命名

在图 3-12 (a) 中，建议读者不要直接采用默认的磁盘容量值（20 GB），需要调大该值，例如设置为 30 GB。如果计算机硬件配置好，可以设置得更大，我们这里设置为 40 GB。单击“下一步”按钮，安装向导显示用户设置的所有信息，如图 3-12 (b) 所示。单击图 3-12 (b) 中的“完成”按钮，进入如图 3-13 所示的界面。



(a) 指定虚拟机磁盘容量



(b) 安装信息小结

图 3-12 指定虚拟机磁盘容量和安装信息小结

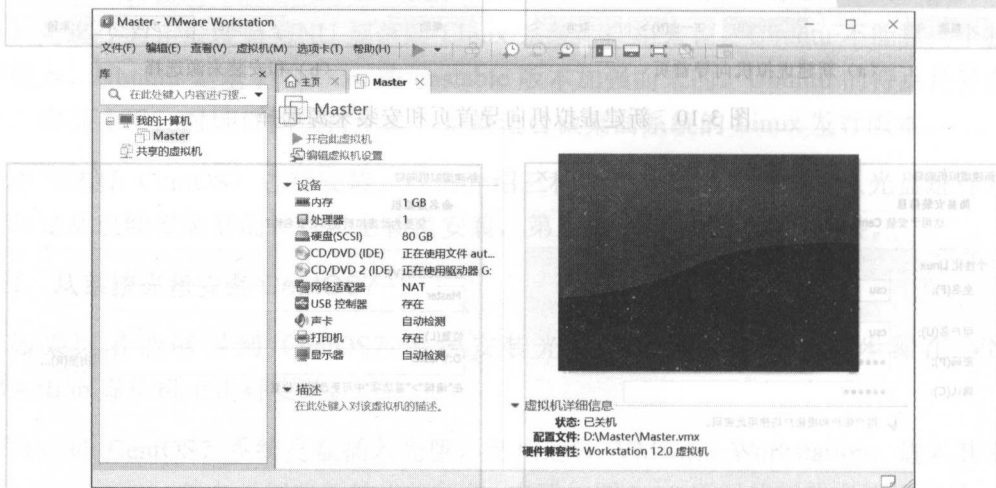


图 3-13 虚拟机设置完成，准备安装 CentOS7

在图 3-13 中，用户可以单击“编辑虚拟机设置”以便修改设备参数，例如，将内存调大到 4 GB。实际上，大数据存储与计算平台往往需要大内存的支持，内存太小会导致系统和应用运行缓慢，甚至无法运行。设置完毕，单击图 3-13 中的“开启此虚拟机”，开始安装 CentOS7，如图 3-14 和图 3-15 所示。

Linux 安装过程自动化程度很高，不需要用户干预，直接等待安装完成即可。安装完毕，光盘自动弹出，系统即进入启动状态，并要求用户输入密码（本书示范设置的密码是 csucsus），如图 3-16 所示。

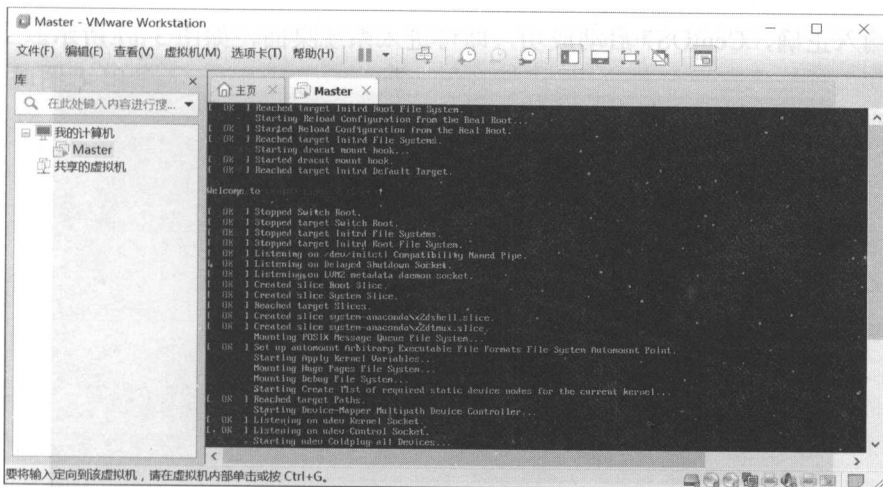


图 3-14 从光盘开始安装 CentOS7

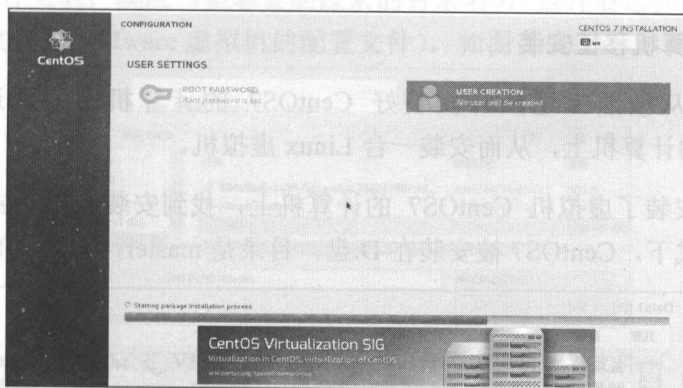


图 3-15 CentOS7 安装中

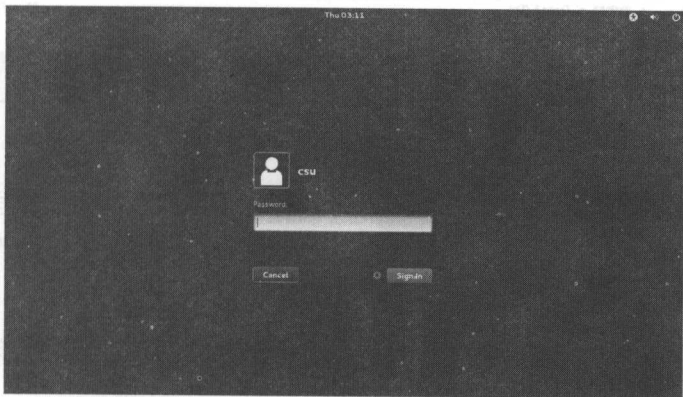


图 3-16 CentOS7 启动要求输入用户密码

密码输入正确，CentOS7 启动成功，即可进入系统桌面，如图 3-17 所示。

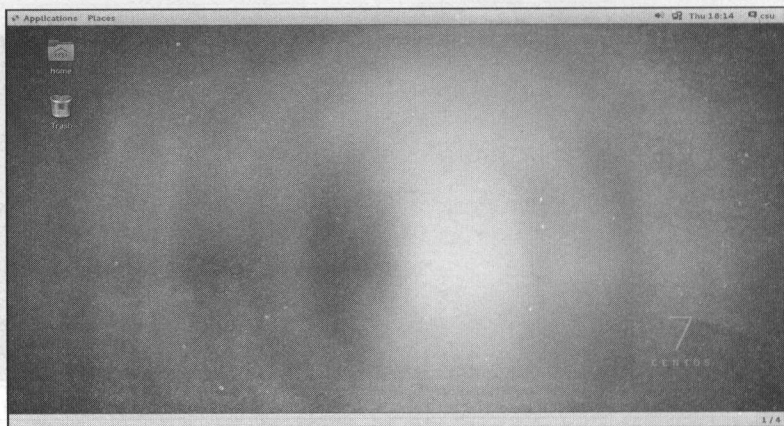


图 3-17 CentOS7 启动成功后进入桌面

2. 从其他计算机移植安装

用户也可以从按照上述步骤安装好 CentOS7 的计算机上，通过简单复制移植 CentOS7 到自己的计算机上，从而安装一台 Linux 虚拟机。

首先，进入安装了虚拟机 CentOS7 的计算机上，找到安装盘和安装目录，例如，本书第一种安装方式下，CentOS7 被安装在 D:盘，目录是 master，如图 3-18 所示。

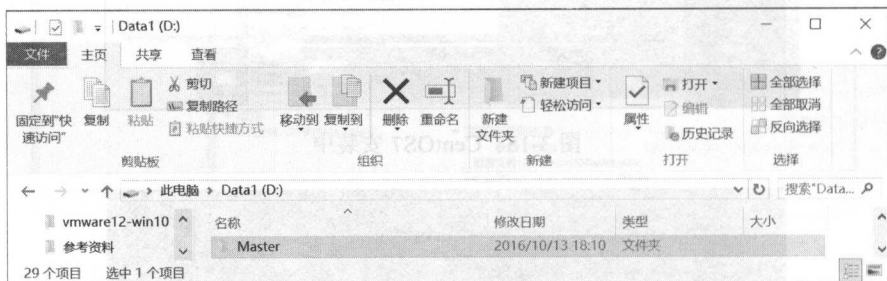


图 3-18 从光盘安装的 CentOS7 所在的驱动器及其安装目录

于是，读者可以将 Master 文件夹复制到自己的计算机硬盘上，这里需要根据自己计算机的情况，将 Master 文件夹放置在合适的驱动器上，并将该文件夹改名为 Slave0（当然，如果您的计算机之前没有安装 Master，也可以不改名）。

特别注意，为了从已经安装好的计算机上克隆 CentOS7，应当先关闭该虚拟机上运行的 CentOS7。

完成上述准备后,启动您计算机上的 VMware Workstation,然后在主页中选择“打开虚拟机”,如图 3-19 所示。

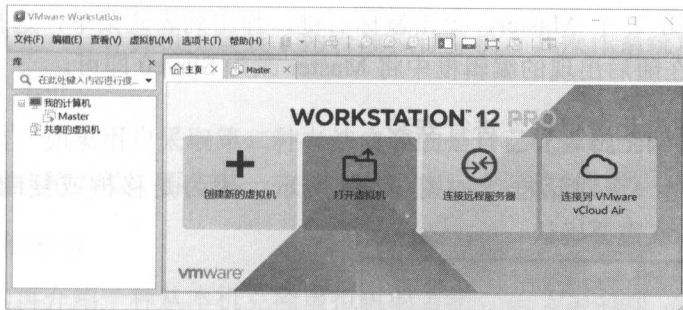


图 3-19 在 VMware Workstation 中选择“打开虚拟机”安装 CentOS7

单击图 3-19 中的“打开虚拟机”后,在弹出的打开文件对话框中,进入 Slave0 目录(如果没有修改文件夹名,就应当是您复制过来的目录名),点开后再选择里面的 master 文件(实际上,该文件是 VMware 虚拟机的配置文件),如图 3-20 所示。

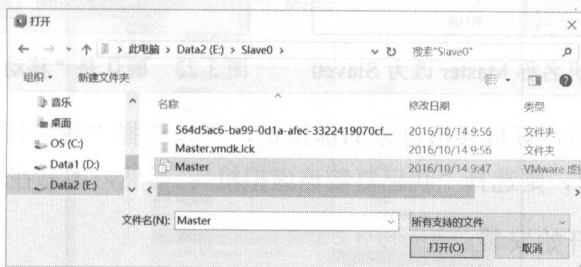


图 3-20 在 VMware Workstation 中打开虚拟机配置文件

单击打开虚拟机配置文件后,进入如图 3-21 所示的界面,表明系统已经准备好开始安装新的虚拟机了。



图 3-21 在 VMware Workstation 中准备安装新的虚拟机 CentOS7

读者注意到,在图 3-21 中,本书是在已经安装了一台虚拟机(CentOS7)的 VMware Workstation 上再安装一台虚拟机 CentOS7,因此需要修改第二台虚拟机的名称。选择“我的计算机”下第 2 个 Master,单击鼠标右键,弹出列表式菜单,选择“重命名(A)”,如图 3-22 所示,在随后出现的编辑框中将 Master 改为 Slave0 即可。

接着单击“开启此虚拟机”进行安装。

这时,会弹出一个对话框,如图 3-23 所示,因为是移植或复制的虚拟机,所以 VMware Workstation 需要确认。

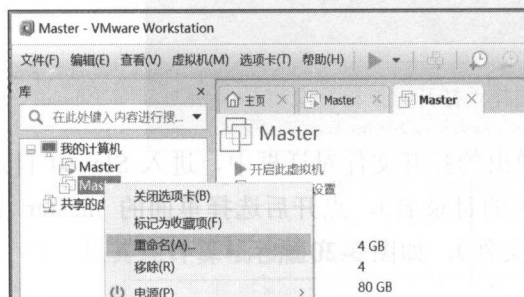


图 3-22 将移植的虚拟机名称 Master 改为 Slave0

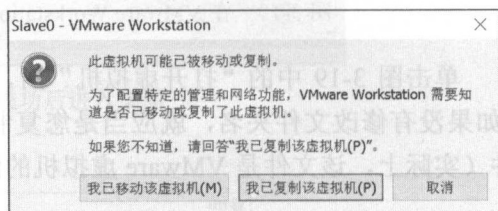


图 3-23 确认是“移动”还是“复制”虚拟机

读者如果是从其他计算机移植过来的虚拟机,可以选择“我已移动该虚拟机”,如果是在本机上复制的虚拟机,则选择“我已复制该虚拟机”。显然,本书在这里应当选择后者。

确认之后,系统开始安装。

同样,安装过程无须用户干预,且安装时间相对较短,直到安装成功,等待用户输入登录密码,如图 3-24 所示。

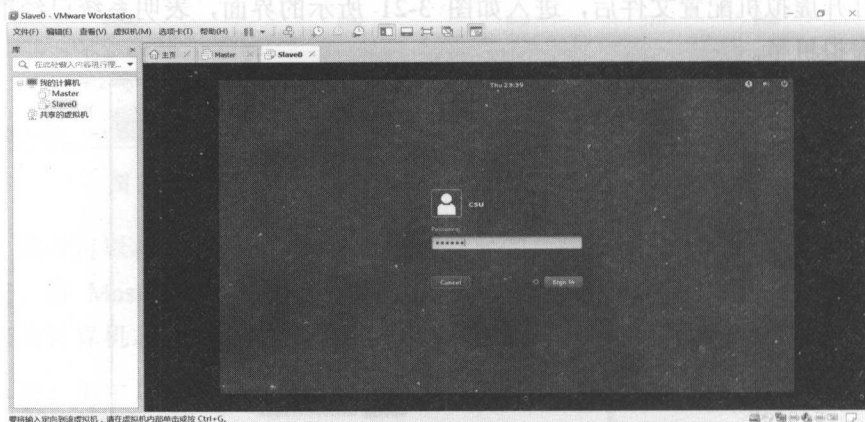


图 3-24 克隆安装虚拟机成功

由于是克隆过来的系统，所以输入的用户密码与原来用户密码是一样的，我们这里是 `csucsu`。

至此，我们就在一台计算机上安装了两台虚拟计算机，其操作系统均为 CentOS7。这实际上是一个最小集群。

值得注意的是，如果用户采取第二种办法安装的是自己计算机上的第一台虚拟机，则可以继续仿照上述移植办法，再安装一台虚拟机。实际上，这种移植安装可以多次进行。

3. 从本机克隆安装

接下来，我们再介绍一种从本机克隆虚拟机的方法。在 VMware Workstation 主界面中，先关闭 Master（注意，克隆前一定要先关闭被克隆虚拟机），选择“我的计算机”下的“Master”，单击鼠标右键，弹出列表式菜单，选择“管理→克隆”，如图 3-25 所示。

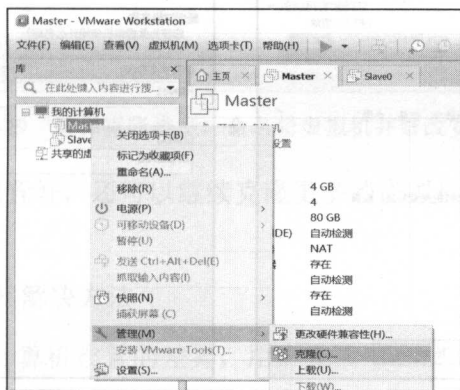


图 3-25 从 Master 直接克隆安装新的虚拟机

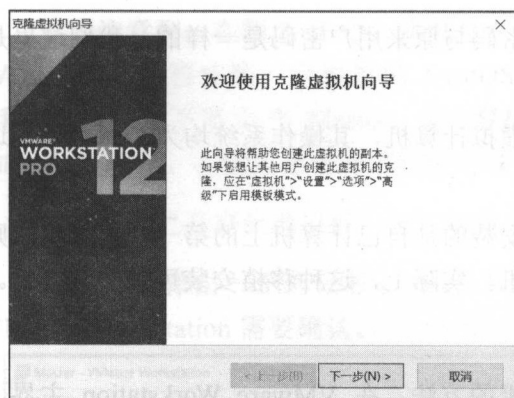
单击图 3-25 中的“克隆”后，VMware 即显示“克隆虚拟机向导”，如图 3-26 (a) 所示。单击“下一步”按钮，选择克隆源，如图 3-26 (b) 所示。

单击图 3-26 (b) 的“下一步”按钮，进入“克隆类型”选择对话框，一般选择“创建完整克隆(F)”，如图 3-27 (a) 所示。

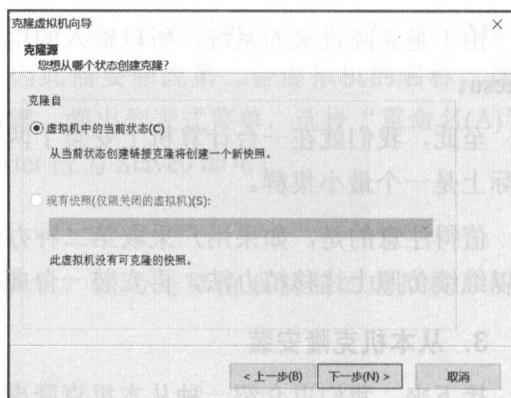
单击图 3-27 (a) “下一步”按钮，进入“新虚拟机名称”对话框，建议将默认的名称和安装位置修改为合适的内容，我们这里给新虚拟机取名为 `Slave1`，安装位置改为 `D:\Slave1`，如图 3-27 (b) 所示，单击“完成”按钮。

单击图 3-27 (a) 中的“完成”按钮后，系统即开始自动安装，这时，只要耐心等待安装完成即可。

安装过程如图 3-28 所示。

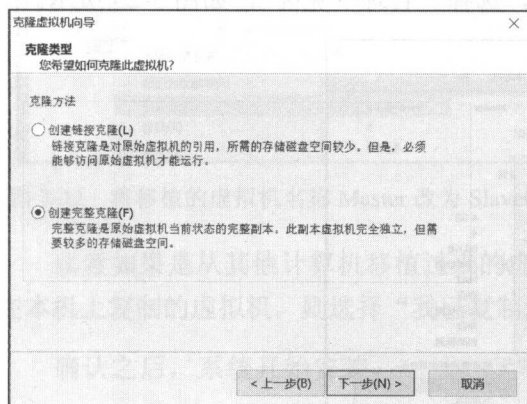


(a)

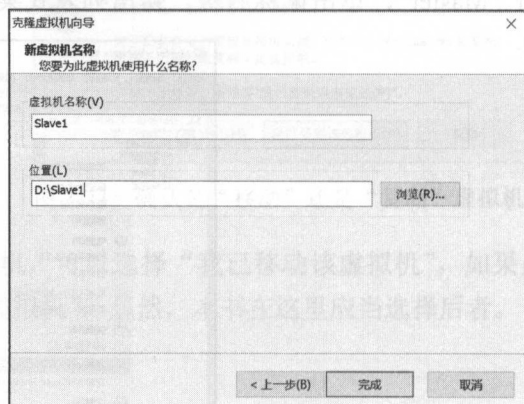


(b)

图 3-26 克隆虚拟机向导

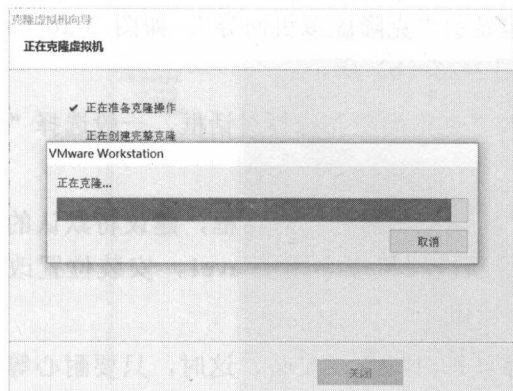


(a)

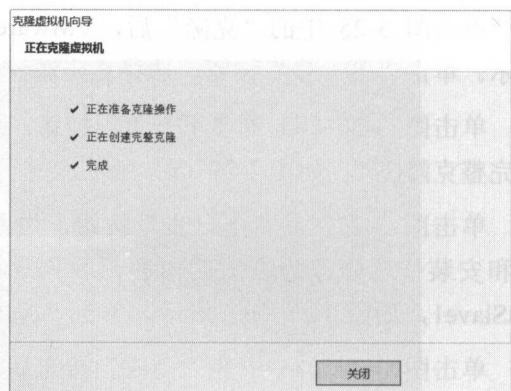


(b)

图 3-27 选择克隆类型，命名新虚拟机并修改安装位置



(a)



(b)

图 3-28 安装进行中

安装完成,单击图 3-28 (a) 中的“关闭”按钮,进入如图 3-29 所示的界面。读者可以发现“我的计算机”列表栏中增加了新安装的“Slave1”。

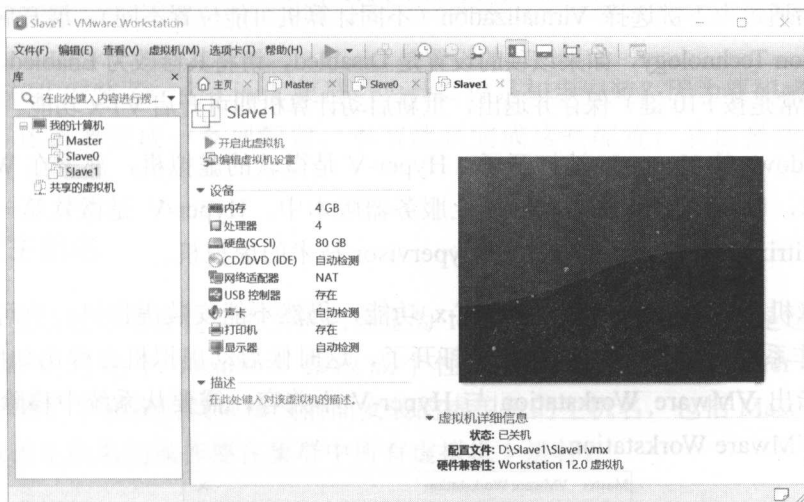


图 3-29 选择克隆类型,命名新虚拟机并修改安装位置

显然,如果计算机配置好,还可以继续克隆更多的虚拟机。但作为学习环境,有 3 台虚拟机就行了。

4. 安装中的问题及其解决方法

由于用户 Windows 计算机设置的差异,用户在 VMware Workstation 上安装 Linux 集群时可能会遇到一些共性问题。

(1) BIOS 中的 VT-x 功能没有打开。如果在安装过程中出现如图 3-30 所示的警告信息框,说明 BIOS 中没有打开 VT-x 功能,所以不能使用 VT-x 进行加速。

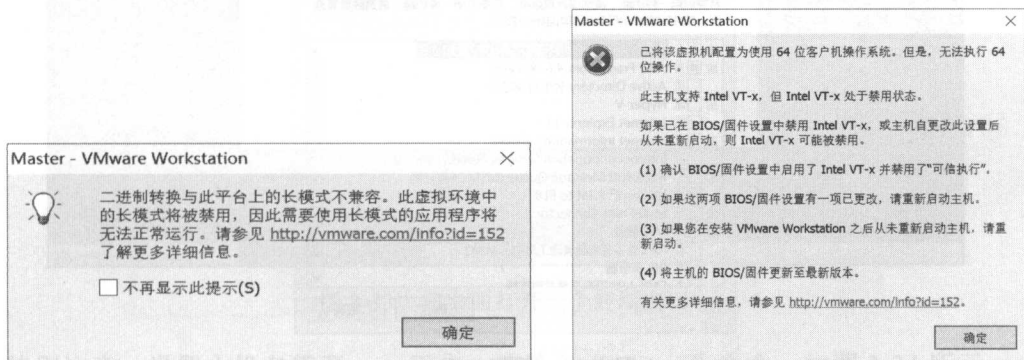


图 3-30 因 BIOS 中没有打开 VT-x 功能引起的警告信息

这时，需要打开 BIOS 中 VT-x 功能。操作如下：首先在开机自检中，按 F12（不同品牌的计算机进入 BIOS 的热键不同，有的计算机是 F1、F2 或 F8）进入 BIOS，找到 Setup 中的 Security，然后通过上下键选择 Virtualization（不同计算机可能位置不同），展开后，出现 Intel (R) Virtualization Technology，如果该项目的设置是 Disabled，请将其修改为 Enabled。修改完毕，按照提示（通常是按 F10 键）保存并退出，重新启动计算机即可开启 VT-x 功能。

(2) Windows 的 Hyper-V 被打开了。Hyper-V 是微软的虚拟机，部署在 Win 8.x 64 位 Pro 以上版本，以及 Windows 2008 以上服务器版本中。Hyper-V 是微软第一个采用类似 VMware 和 Citrix 开源 Xen 一样的基于 hypervisor 技术的虚拟机。

如果计算机即使打开了 BIOS 中 VT-x 功能，仍然不能安装虚拟机，有可能是由于其 Windows 操作系统的 Hyper-V 功能已经打开了，这时候启动虚拟机会弹出如图 3-31 所示的提示框，指出 VMware Workstation 与 Hyper-V 不兼容，需要从系统中移除 Hyper-V 角色，然后再运行 VMware Workstation。

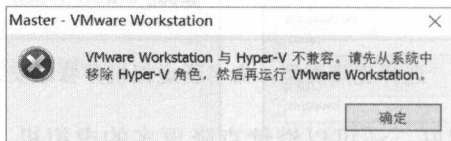


图 3-31 VMware Workstation 与 Hyper-V 不兼容提示

以 Win10 为例，为了关闭 Hyper-V，首先右键单击 Windows 左下角徽标，选择“程序和功能(F)”，在打开的“程序与功能”对话框中，单击“启用或关闭 Windows 功能”，弹出如图 3-32 所示的页面，找到 Hyper-V 选择项，将其设置为关闭，即把“√”去掉，然后单击“确定”即可。



图 3-32 关闭 Hyper-V

3.3 集群的配置

上面我们初步安装了一个 Linux 集群。为什么说是初步安装？因为要想将上述集群投入实际运行，还需要完成一系列配置。本节就来完成这些配置，从而搭建一个可用的 Linux 集群。

3.3.1 设置主机名

CentOS7 安装后系统会自动将主机名确定为 localhost，初装的主机都是这样，因此，用户一般都希望将这个 localhost 修改成个性化的主机名，如 Master、Slave0 或 HadoopMaster 等。下面我们就来修改前面安装的虚拟机的主机名，包括 Master 和 Slave，也就是说，修改主机名的操作要在集群中所有虚拟机上进行。

启动所有虚拟机。

我们示范在 Master 主机上的操作。在其桌面开一个终端窗口，方法是单击左上角的“Applications”→“Utilities”→“Terminal”；或者在桌面任意位置单击鼠标右键，在弹出的列表中选择“Open in Terminal”，打开的终端如图 3-33 所示。

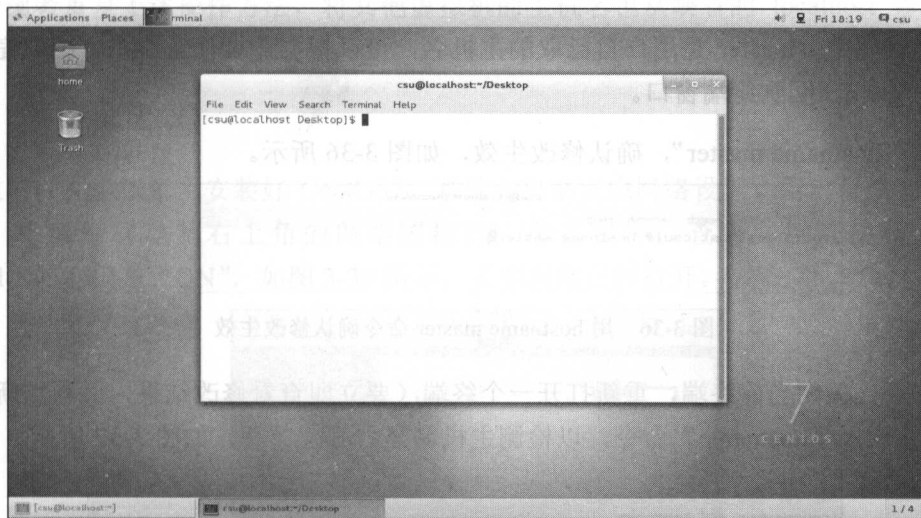


图 3-33 在桌面打开一个终端窗口

特别注意，设置主机名需要 root 用户，请输入“su root”命令，如图 3-34 所示。

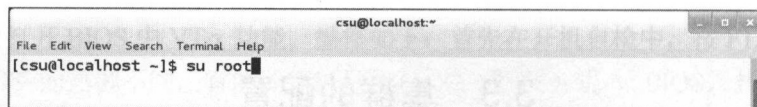


图 3-34 切换到 root 用户的命令

回车后输入密码（这里是 csucsu），切换到 root 用户。注意，这时候的提示符是“#”，这是 Linux 表示 root 用户的命令行提示符，而一般用户的提示符是“\$”。

接下来使用 gedit 编辑主机名。如果不使用 gedit 编辑器（或者由于某种故障不能使用），也可以采用 vi 编辑器（后面用到 gedit 的地方都可以用 vi 代替）。

输入“gedit /etc/sysconfig/network”命令，准备编辑 network 配置文件，如图 3-35 所示。

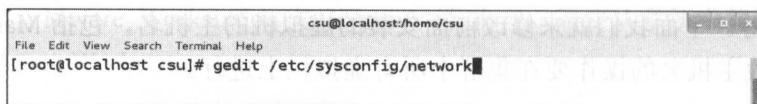


图 3-35 编辑 networks 配置文件的命令

在打开的 gedit 编辑器中，输入如下代码。

```

NETWORKING=yes
HOSTNAME=master

```

上述代码中，master 是用户自己取的主机名，可以根据需要任意命名。编辑完毕，保存并退出 gedit，回到终端窗口。

输入“hostname master”，确认修改生效，如图 3-36 所示。

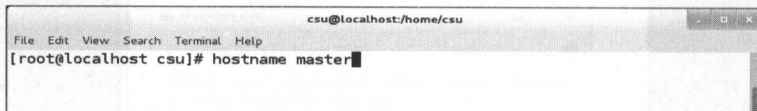


图 3-36 用 hostname master 命令确认修改生效

回车后，关闭当前终端，重新打开一个终端（要立即查看修改结果，必须重新开一个终端），并输入“hostname”命令，以检测主机名是否修改成功，如图 3-37 所示。

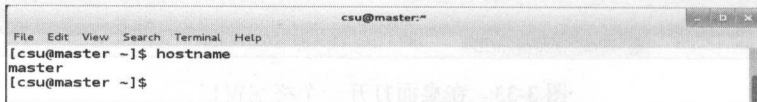


图 3-37 用 hostname 命令检测修改是否成功

可以看到, 不仅命令的返回值正是用户所取的 `master` 主机名, 而且提示符内的主机名称也由 `localhost` 改为 `master` 了。

但是, 如果只修改 `network` 文件, 下次重启虚拟机时, 会发现修改后的名字又回到原来的 `localhost` 去了。原来上面只修改了瞬态 (Transient) 主机名, 并没有修改静态 (Static) 主机名。

因此, 仍然要使用 `root` 用户进行修改, 但这一次输入 “`gedit /etc/hostname`” 命令, 如图 3-38 所示。

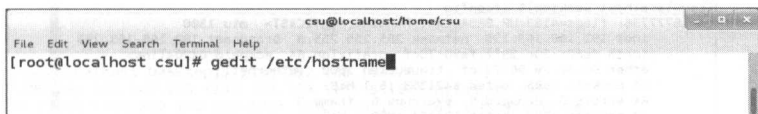


图 3-38 用 `gedit /etc/hostname` 命令修改 `hostname` 文件

在编辑器中输入如下代码:

```
master
```

上述代码中, `Master` 是输入的新主机名。其实, 原来文件里有一个 `localhost.localdomain` 的主机名, 应当将其删除掉, 保存后退出。这样, 重启虚拟机后, 主机名就永久修改了。

请读者重复上述操作方法, 将其他虚拟机的主机名也从缺省的 `localhost` 修改为对应的 `slave0` 和 `slave1` (如果安装了)。

3.3.2 网络设置

通常, 在虚拟机上安装好 `CentOS7`, 系统会自动完成网络设置。用户可以在虚拟机桌面上, 将鼠标移动到右上角的网络图标处, 单击鼠标, 查看网络连接状态, 如果 “`Wired`” 的标识是 “`ON`”, 如图 3-39 所示, 表明网络已经打开。

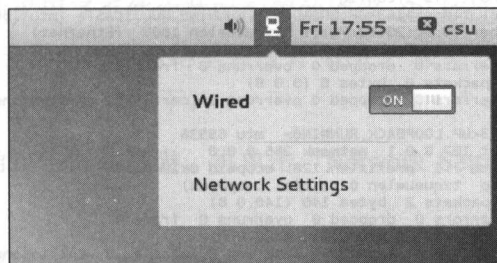


图 3-39 `CentOS7` 桌面右上角的网络图标

然后，在终端中输入“ifconfig”命令，如图 3-40 所示，以查看 Master 的 IP 地址。

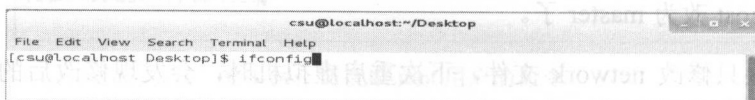


图 3-40 用 ifconfig 命令查看 Master 的 IP 地址

回车后，终端窗口中将显示 Master 的 IP 地址配置信息，如图 3-41 所示。

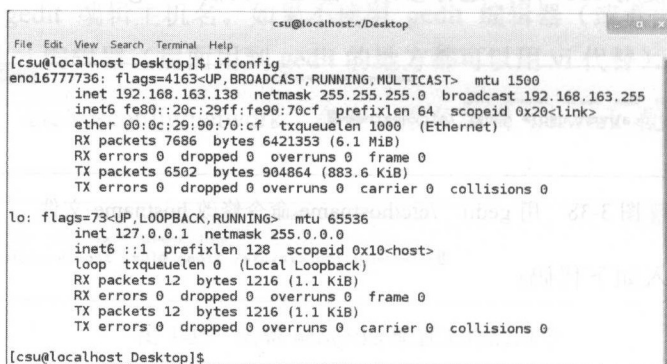


图 3-41 终端显示 Master 的 IP 地址配置信息

可以看到，安装过程中，系统将 Master 的 IP 地址自动配置为 192.168.163.138（读者计算机的配置可能不一样），此外还配置了广播地址、子网掩码等。如果情况是这样，说明该节点的网络配置已经就绪，可以开始使用网络功能了，例如，启动浏览器访问内部或外部网站。

如果用 ifconfig 命令看到的显示信息类似如图 3-42 所示的样子，表明网络连接没有打开或配置没有完成，则需要打开网络连接或进行网络配置。

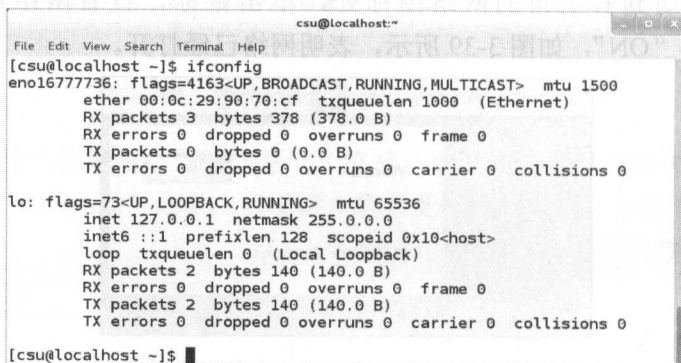


图 3-42 网络没有打开或配置没有完成

要打开网络，请单击桌面右上角的网络图标（如图 3-39 所示），用鼠标将灰色的 OFF 状态改为彩色的 ON 状态即可。

注意，上述设置也必须在 Slave0 和 Slave1 进行，并确保集群的每一台虚拟机都在同一个网段内，例如 Slave0 的 IP 地址可以是 192.168.163.135，Slave1 的 IP 地址可以是 192.168.163.137。

这样，我们就可以在终端中用 ping 命令测试集群是否能够连通。图 3-43 是在 Master 上用 ping 命令测试 Slave0 的显示，如果看到有返回值，说明两台虚拟机是连通的。

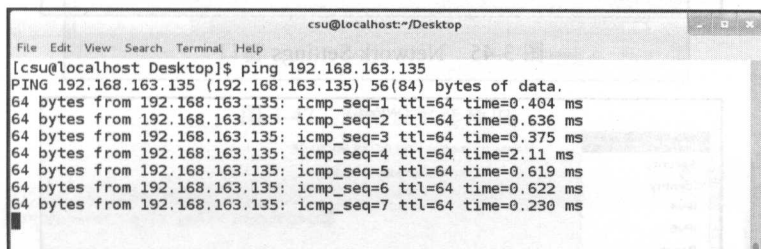


图 3-43 Master 可以 ping 通 Slave0

如果看到的是如图 3-44 所示的情况，说明网络连接失败，需要打开网络或重新配置。

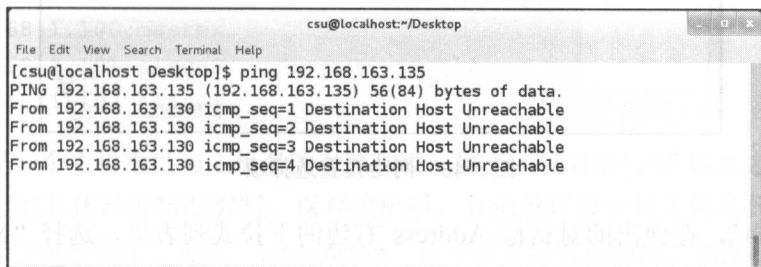


图 3-44 Master 不能 ping 通 Slave0

上面我们只是简单地采用了系统安装时配置的网络 IP 地址。实际上，上述地址一般都是 DHCP 类型的。由于我们目前网络里并没有 DHCP 服务器，因此，需要将动态主机 IP 地址修改为静态地址。具体操作如下。

在虚拟机桌面右上角单击网络图表，再单击“Network Settings”，弹出如图 3-45 所示的窗口。

选择“Wired”，并将“ON”状态改为“OFF”。注意，修改网络地址需要事先断开网络连接。然后单击图 3-44 右下角的设置图标，弹出如图 3-46 所示的窗口。

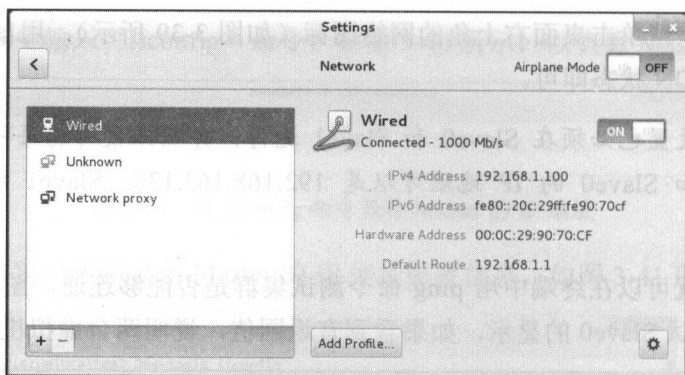


图 3-45 Network Settings 窗口

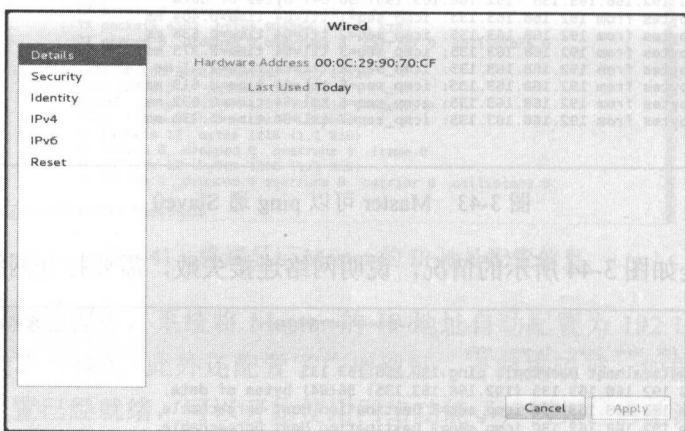


图 3-46 网络设置选择项

选择“IPv4”，在弹出的对话框 Address 右边的下拉式列表里，选择“Manual”，然后在编辑框中输入静态地址信息，如图 3-47 所示。这里，我们设置的地址是 192.168.1.100，子网掩码是 255.255.255.0，网关地址是 192.168.1.1。读者可以根据自己需要设置。设置完毕，单击“Apply”按钮退出即可。

按照上述方法，将 Slave 的 IP 地址也修改过来。例如，Slave0 是 192.168.1.101，Slave1 是 192.168.1.102，网关一样。这样，我们就完成了整个集群的网络配置。读者可以用 ping 命令测试计算机之间的连通性。

然而，为了用计算机名进行网络访问，我们还需要修改 hosts 文件中的主机名与 IP 地址对照列表。注意，仍然需要在 root 用户下进行操作。输入“gedit /etc/hosts”，如图 3-48 所示。

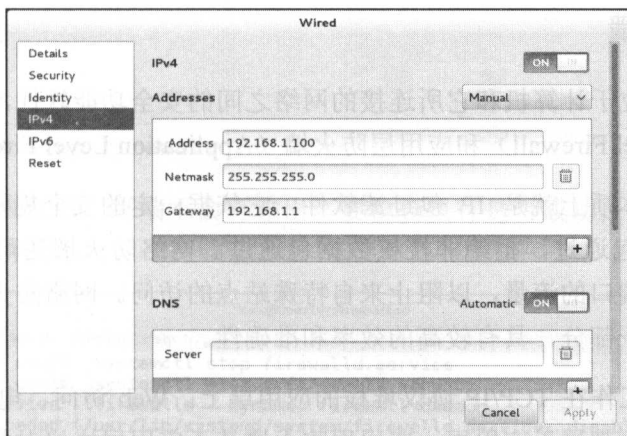


图 3-47 网络设置选择项

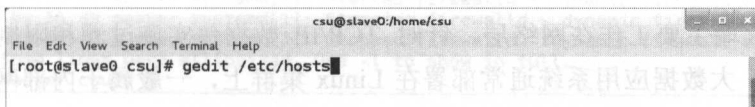


图 3-48 修改主机列表文件

输入如下代码（注意，保留文件中已有代码，在后面添加新代码）：

```
192.168.1.100 master
192.168.1.101 slave0
192.168.1.102 slave1
```

上述代码是一个主机地址与主机名的对照列表，用于主机名称与 IP 地址之间的解析。读者可以根据自己的具体设置情况编写。保存退出后，我们就可以使用主机名代替其 IP 地址了，例如在任意主机终端中，通过“ping master”命令即可测试连通性，如图 3-49 所示。

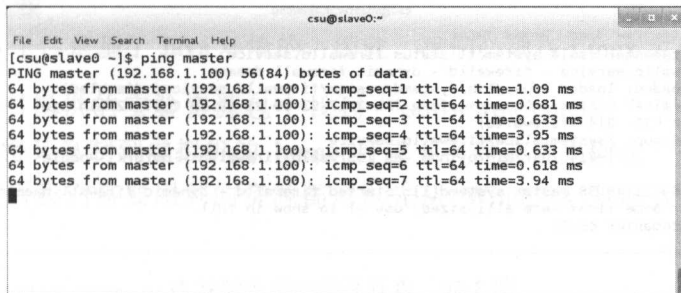


图 3-49 利用主机名测试网络连通性

注意，上述修改需要在集群的所有主机上进行。

3.3.3 关闭防火墙

防火墙是一种位于计算机和它所连接的网络之间的安全功能软件，一般分为网络层防火墙（Network Level Firewall）和应用层防火墙（Application Level Firewall）两种。

网络层防火墙本质上就是 IP 包过滤软件，它依据一定的安全规则，对 IP 包进行检测，允许授权数据包通过，拒绝非授权数据包通过。网络防火墙还可以关闭不使用的端口，或者禁止特定端口的流量，以阻止来自特殊站点的访问。网络防火墙处于操作系统的网络层，是内核的一部分，具有较高的效率和准确性。

应用层防火墙工作在 TCP/IP 协议堆栈的应用层上。Web 访问、电子邮件和 FTP 文件传输所产生的数据流都属于应用层。应用层防火墙基于应用层协议对数据进行扫描和安全处理，通常会直接丢弃不安全的数据包。

Linux 防火墙主要工作在网络层，针对 TCP/IP 数据包实施过滤和限制，属于典型的包过滤防火墙。大数据应用系统通常部署在 Linux 集群上，一般属于内部网络平台，且计算机之间关系十分密切，通信频繁，因此不需要启用防火墙。

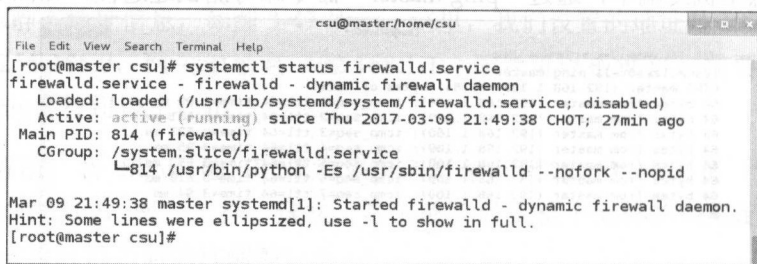
下面我们来关闭集群的防火墙（请读者注意，CentOS7 关闭防火墙的操作命令已经不同于早期版）；以下的操作在 root 账号下进行，而且需要在所有计算机上实施。

1. 检查系统防火墙状态

要查看系统防火墙的当前运行状态，可使用如下命令：

```
systemctl status firewalld.service
```

图 3-50 给出了命令的输入和执行输出。



```
csu@master:~/home/csu
File Edit View Search Terminal Help
[root@master csu]# systemctl status firewalld.service
firewalld.service - firewalld - dynamic firewall daemon
   Loaded: loaded (/usr/lib/systemd/system/firewalld.service; disabled)
   Active: active (running) since Thu 2017-03-09 21:49:38 CHOT; 27min ago
     Main PID: 814 (firewalld)
    CGroup: /system.slice/firewalld.service
            └─814 /usr/bin/python -Es /usr/sbin/firewalld --nofork --nopid

Mar 09 21:49:38 master systemd[1]: Started firewalld - dynamic firewall daemon.
Hint: Some lines were ellipsized, use -l to show in full.
[root@master csu]#
```

图 3-50 查看系统的防火墙状态

在图中我们看到 Master 计算机上的防火墙服务处于活动状态（Active），可见初次安装 Linux 后系统会默认启用防火墙。下面我们将防火墙关闭掉。

2. 关闭防火墙

使用的命令是：

```
systemctl stop firewalld.service
```

该命令输入和执行后，终端不会有输出。要检查执行是否成功，可以再次使用状态检查命令，如图 3-51 所示。

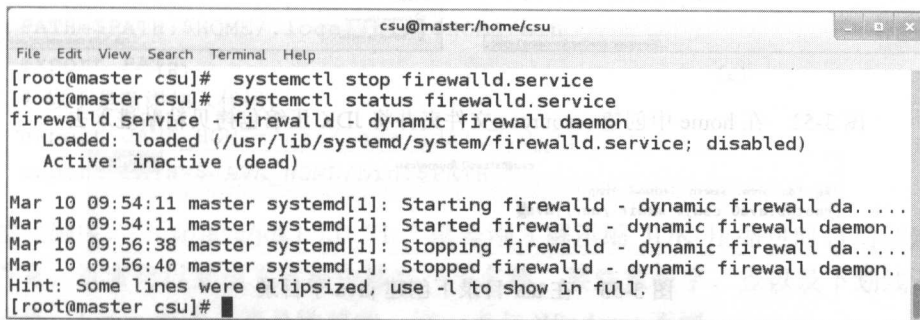


图 3-51 防火墙被关闭后的状态信息

一般在关闭防火墙之后，还可以执行下面的命令：

```
systemctl disable firewalld.service
```

该命令使防火墙在下次启动计算机的时候取消防火墙服务。

读者注意，上述各项操作需要在所有计算机上进行。这样，我们就完成了关闭集群防火墙的工作，从而为后面安装 Hadoop 并运行 MapReduce 程序创造了条件。

3.3.4 安装 JDK

由于 Hadoop 平台是基于 Java 环境的，因此，我们必须在 Linux 主机上安装 JDK。

请读者在本书提供的实验资源包中找到“\software\CentOS7”下的 jdk-7u71-linux-x64.gz 压缩包，将其复制到虚拟机的 home 目录下（建议首先创建一个新文件夹，命名为 resources，以后我们需要用到的资源都可以先从 Windows 复制到这里），如图 3-52 所示。

1. 在 root 账号下安装 JDK

打开终端，切换成 root 用户（注意，安装 JDK 必须使用 root 用户权限）。在 usr 目录下创建 java 子目录，即输入“mk_dir/usr/java 命令”，如图 3-53 所示。

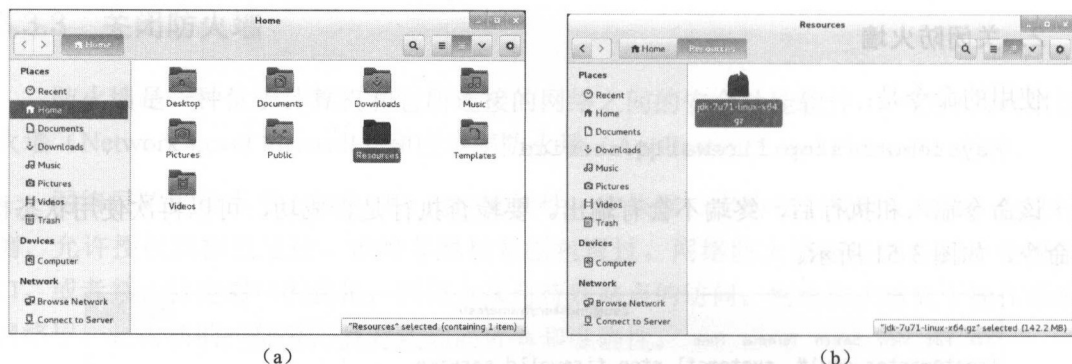


图 3-52 在 home 中创建 resources 文件夹并将 JDK 压缩包拷贝粘贴进去

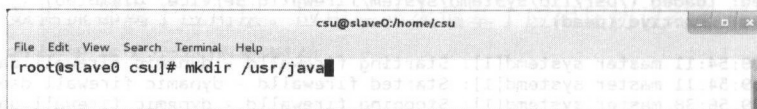


图 3-53 在 usr 目录下创建 java 子目录

然后执行“`mv /home/csu/resources/jdk-7u71-linux-x64.gz /usr/java`”命令，将 JDK 压缩包复制到上面创建的“`/usr/java`”目录下，如图 3-54 所示。

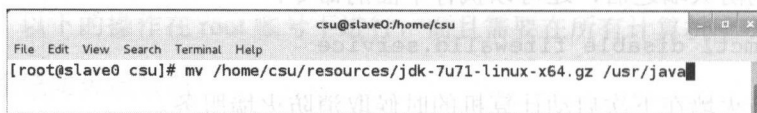


图 3-54 将 JDK 压缩包移动到 java 子目录下

执行“`cd /usr/java`”命令以便切换到 java 目录下。

解压缩 JDK 压缩文件，即执行“`tar -zxvf/usr/java/jdk-7u71-linux-x64.gz`”命令，如图 3-55 所示。

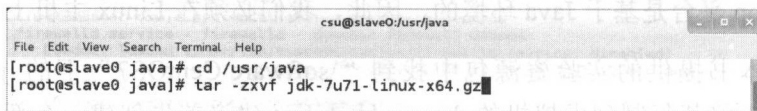


图 3-55 准备解压缩 JDK 文件

回车后系统开始执行解压缩命令，读者可以看到屏幕上不断滚动显示的信息。执行成功后，我们还需要配置 Java 环境变量。

2. 配置环境变量

执行“`gedit /home/csu/.bash_profile`”命令，如图 3-56 所示。

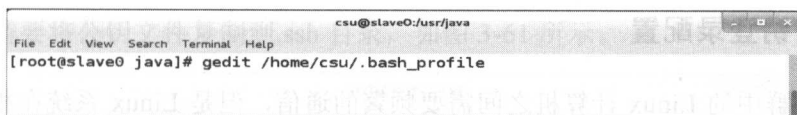


图 3-56 编辑 Java 环境变量

在打开的文件中，在已有代码的尾部添加如下代码（已有代码可以不动）：

```
# User specific environment and startup programs
PATH=$PATH:$HOME/.local/bin:$HOME/bin
export PATH
# 以下是新添加代码：
export JAVA_HOME=/usr/java/jdk1.7.0_71/
export PATH=$JAVA_HOME/bin:$PATH
```

上述代码中，“/usr/java/jdk1.7.0_71/”就是刚才解压缩 JDK 压缩包时自动创建的 java 安装文件夹，读者可以到该文件夹中核实一下名称，要注意数字、点以及下划线不要输入错误；此外，Linux 对大小写是敏感的，这一点与 Windows 不同。

接着，执行“source /home/csu/.bash_profile”命令，使修改生效，如图 3-57 所示。

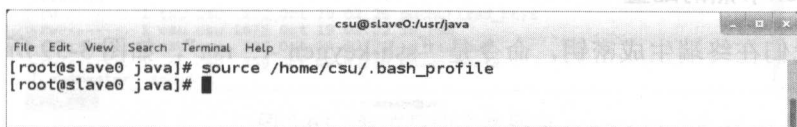


图 3-57 用 source 命令使修改生效

如果没有编辑错误，系统执行 source 命令成功后没有任何返回信息，最后执行“java-version”测试配置是否成功，如图 3-58 所示。

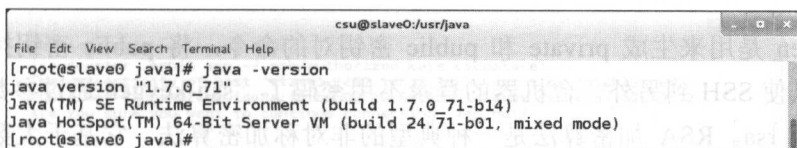


图 3-58 测试 JDK 安装是否成功

如果出现图 3-58 所显示的版本信息、运行时环境，说明 JDK 安装成功。

注意，上述安装和配置还需要在集群的其他计算机上进行。这里读者会提出一个问题，就是如果集群有成千上万台计算机，难道也要这样一台一台地去安装和配置吗？当然不需要这样重复安装。实际上，只要先安装好一台计算机，其他的计算机可以复制。

3.3.5 免密钥登录配置

大数据集群中的 Linux 计算机之间需要频繁的通信,但是 Linux 系统在相互通信中需要进行用户身份认证,也就是输入登录密码。在集群规模不大的情况下,每次登录少量计算机进行输入密码认证,所需要的操作时间尚且不多。但是,如果集群是几十台、上百台甚至上千台计算机,频繁的认证操作就会大大降低工作效率,因此,实际生产中的集群都需要进行免密钥登录配置。

免密钥登录是指两台 Linux 机器之间使用 SSH 连接时不需要用户名和密码。SSH (Secure Shell Protocol, 安全外壳协议) 是一种在不安全网络上提供安全远程登录及其他安全网络服务的协议。默认状态下,SSH 连接是需要密码认证的,但是可以通过修改系统认证,使系统通信免除密码输入和 SSH 认证。

下面的配置分为在 Master 节点的操作和在所有 Slave 节点上的操作。同时注意,这里使用普通账户(本书是 csu),所以如果目前处于 root 用户状态,则需要切换回普通用户状态。

1. Master 节点的配置

首先,我们在终端生成密钥,命令是“ssh-keygen -t rsa”,如图 3-59 所示。

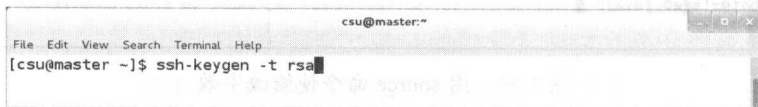


图 3-59 生成密钥的命令

回车后,系统会出现一系列提示,这时候只要简单回车即可。

ssh-keygen 是用来生成 private 和 public 密钥对的命令,将 public 密钥拷贝到远程机器后,就可以使 SSH 到另外一台机器的登录不用密码了。ssh-keygen 通过参数-t 指定加密算法,这里是 rsa。RSA 加密算法是一种典型的非对称加密算法,它基于大数的因式分解数学难题,是应用最广泛的非对称加密算法。

生成的密钥在.ssh目录下,可以切换到该目录用“ls -l”命令查看,如图 3-60 所示。

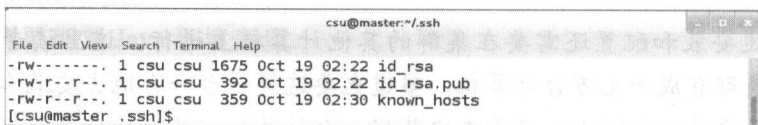


图 3-60 查看生成的密钥文件

接下来需要将公钥文件复制到.ssh 目录, 如图 3-61 所示。

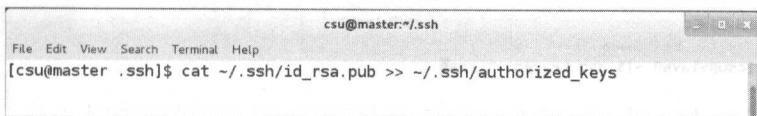


图 3-61 复制公钥文件

复制过来是为了便于修改该文件的权限, 请用“`chmod 600 ~/.ssh/authorized_keys`”命令进行修改, 如图 3-62 所示。

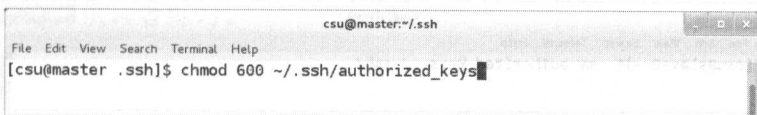


图 3-62 修改公钥文件权限

修改权限后, 文件列表情况如图 3-63 所示。

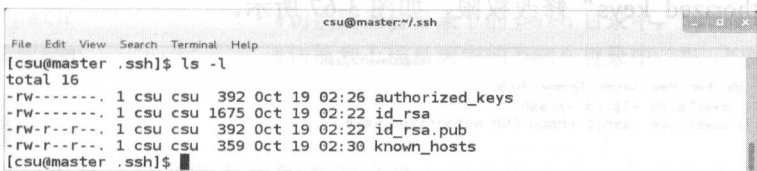


图 3-63 修改后的文件列表

最后将 `authorized_keys` 文件复制到所有的 Slave 节点。我们这里需要分别复制到 Slave0 和 Slave1 节点上, 命令是“`scp ~/.ssh/authorized_keys csu@slave0:~/`”和“`scp ~/.ssh/authorized_keys csu@slave1:~/`”, 如图 3-64 所示。

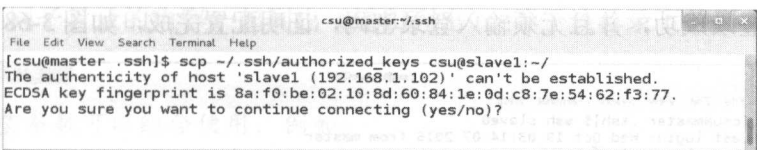


图 3-64 将 `authorized_keys` 文件复制到 slave1 节点

如果出现如图 3-63 所示的提示, 输入“yes”后回车, 然后输入密码(我们这里是 csucsu)。至此, 就完成了在 Master 节点上的配置。

2. Slave 节点的配置

完成 Master 节点的配置后, 需要转到 Slave 节点进行配置。

首先，还是用“ssh-keygen”命令生成密钥，如图 3-65 所示，一路回车即可。

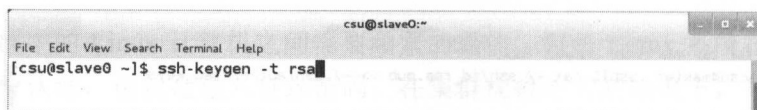


图 3-65 在 Slave0 节点生成非对称密钥

接着将 authorized_keys（注意，这个文件是从 Master 复制过来的）文件移动到.ssh 目录下，命令是“mv authorized_keys ~/.ssh”如图 3-66 所示。

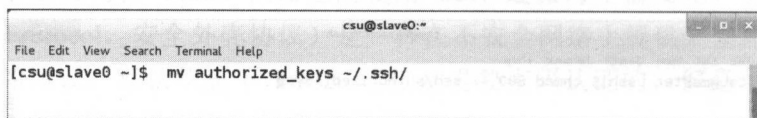


图 3-66 移动 authorized_keys 文件

最后，修改 authorized_keys 文件的权限。先用 cd 命令切换到.ssh 目录，再输入命令“chmod 600 authorized_keys”修改权限，如图 3-67 所示。

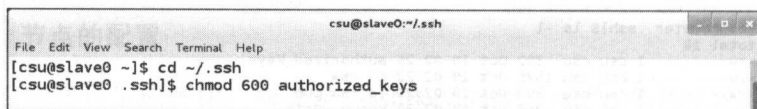


图 3-67 修改 authorized_keys 文件的权限

上述配置也需要在 Slave1（如果安装了）上做一遍。

至此，免登录密钥配置进行完毕。

下面来验证一下配置是否有效。在 Master 机器上执行“ssh slave0”命令，登录 Slave0 计算机，如果登录成功，并且无须输入登录密码，证明配置完成，如图 3-68 所示。

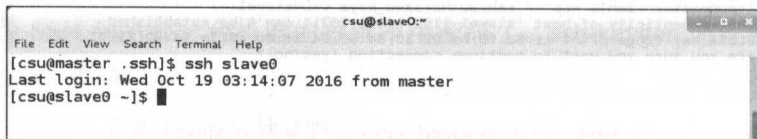


图 3-68 验证免密钥登录配置

从图 3-64 可以看出，Master 通过 ssh 登录到 Slave0 计算机。因此，可以在 Master 上的终端操作 Slave0 计算机了，这样就不用走到 Slave0 计算机前进行操作，可节省工作时间。

要退出远程节点回到本地计算机，只要输入 exit 命令即可。

3.4 Linux 基本命令

实际上，我们在上面已经使用了一些常见的 Linux 命令，如 ls、cd、mkdir、cat、mv、su、ifconfig、ping、tar、source 等。Linux 命令分为两大类，一类叫内部命令，是指由 Linux shell 实现的命令；另一类是外部命令，指通过外部程序提供的命令，如 java、javac 等。

要熟练使用 Linux 操作系统，掌握各种命令的用法是基本要求。我们在这里列出一些常见的 Linux 命令，供读者练习。

1. cd 命令

这是一个非常基本，也是大家经常需要使用的命令，用于改变文件目录，它的参数是要切换到目录的路径，可以是绝对路径，也可以是相对路径，例如：

```
cd /root/Documents      #切换到目录/root/Documents
cd ./path                #切换到当前目录下的 path 目录中，"."表示当前目录
cd ../path               #切换到上层目录中的 path 目录中，".."表示上一层目录
```

2. ls 命令

即 list 之意，也是一个非常有用的查看文件与目录的命令，它的参数非常多，如

- -l: 列出长数据串，包含文件的属性与权限数据等。
- -a: 列出全部的文件，连同隐藏文件（开头为.的文件）一起列出来（常用）。
- -d: 仅列出目录本身，而不是列出目录的文件数据。
- -h: 将文件容量以较易读的方式（GB、KB 等）列出来。
- -R: 连同子目录的内容一起列出（递归列出），等于该目录下的所有文件都会显示出来。

注意，这些参数可以组合使用，例如：

```
ls -l                #以长数据串的形式列出当前目录下的数据文件和目录
ls -lR               #以长数据串的形式列出当前目录下的所有文件
```

3. grep 命令

该命令常用于分析一行的信息，如果当中有我们所需要的信息，就将该行显示出来，该命令通常与管道命令一起使用，用于对一些命令的输出进行筛选加工等。

它的常用参数如下。

- **-a:** 将 binary 文件以 text 文件的方式查找数据。
- **-c:** 计算找到“查找字符串”的次数。
- **-i:** 忽略大小写的区别, 即把大小写视为相同。
- **-v:** 反向选择, 即显示出没有“查找字符串”内容的那一行。

例如:

```
grep --color=auto 'MANPATH' /etc/man.config #取出文件/etc/man.config
中包含 MANPATH 的行, 并把找到的关键字加上颜色
ls -l | grep -i file #把 ls -l 的输出中包含字母 file (不区分大小写) 的
内容输出
```

4. find 命令

find 是一个基于查找的功能非常强大的命令, 相对而言, 它的使用也相对复杂, 参数比较多。它的基本语法如下:

```
find [PATH] [option] [action]
```

(1) 与时间有关的参数。

- **-mtime n:** *n* 为数字, 意思为在 *n* 天之前的“一天内”被更改过的文件。
- **-mtime +n:** 列出在 *n* 天之前 (不含 *n* 天本身) 被更改过的文件名。
- **-mtime -n:** 列出在 *n* 天之内 (含 *n* 天本身) 被更改过的文件名。
- **-newer file:** 列出比 *file* 还要新的文件名。

例如:

```
find /root -mtime 0 #在当前目录下查找今天之内有改动的文件
```

(2) 与用户或用户组名有关的参数。

- **-user name:** 列出文件所有者为 *name* 的文件。
- **-group name:** 列出文件所属用户组为 *name* 的文件。
- **-uid n:** 列出文件所有者为用户 ID 为 *n* 的文件。
- **-gid n:** 列出文件所属用户组为用户组 ID 为 *n* 的文件。

例如:

```
find /home/csu -user ljianhui #在目录/home/csu 中找出所有者为 csu 的文件
```

(3) 与文件权限及名称有关的参数。

- **-name filename:** 找出文件名为 *filename* 的文件。

- `-size [+|-]SIZE`: 找出比 SIZE 还要大 (+) 或小 (-) 的文件。
- `-type TYPE`: 查找文件的类型为 TYPE 的文件, TYPE 的值主要有一般文件 (f)、设备文件 (b、c)、目录 (d)、连接文件 (l)、socket (s)、FIFO 管道文件 (p)。
- `-perm mode`: 查找文件权限刚好等于 mode 的文件, mode 用数字表示, 如 0755。
- `-perm -mode`: 查找文件权限必须要全部包括 mode 权限的文件, mode 用数字表示。
- `-perm +mode`: 查找文件权限包含任一 mode 的权限的文件, mode 用数字表示。

例如:

```
find / -name passwd      #查找文件名为 passwd 的文件
find . -perm 0755        #查找当前目录中文件权限的 0755 的文件
find . -size +12k        #查找当前目录中大于 12KB 的文件, 注意 c 表示 byte
```

5. cp 命令

该命令用于复制文件, copy 之意, 它还可以把多个文件一次性地复制到一个目录下, 它的常用参数如下。

- `-a`: 将文件的特性一起复制。
- `-p`: 连同文件的属性一起复制, 而非使用默认方式, 与 `-a` 相似, 常用于备份。
- `-i`: 若目标文件已经存在时, 在覆盖时会先询问操作的进行。
- `-r`: 递归持续复制, 用于目录的复制行为。
- `-u`: 目标文件与源文件有差异时才会复制。

例如:

```
cp -a file1 file2        #连同文件的所有特性把文件 file1 复制成文件 file2
cp file1 file2 file3 dir  #把文件 file1、file2、file3 复制到目录 dir 中
```

6. mv 命令

该命令用于移动文件、目录或更名, move 之意, 它的常用参数如下。

- `-f`: force, 强制的意思, 如果目标文件已经存在, 不会询问而直接覆盖。
- `-i`: 若目标文件已经存在, 就会询问是否覆盖。
- `-u`: 若目标文件已经存在, 且比目标文件新, 才会更新。

注意, 该命令可以把一个文件或多个文件一次移动到一个文件夹中, 但是最后一个目标文件一定要是“目录”。

例如:

```
mv file1 file2 file3 dir #把文件 file1、file2、file3 移动到目录 dir 中
mv file1 file2           #把文件 file1 重命名为 file2
```

7. rm 命令

该命令用于删除文件或目录，**remove** 的意思，它的常用参数如下。

- **-f**: 就是 **force** 的意思，忽略不存在的文件，不会出现警告消息。
- **-i**: 互动模式，在删除前会询问用户是否操作。
- **-r**: 递归删除，最常用于目录删除，它是一个非常危险的参数。

例如：

```
rm -i file           #删除文件 file，在删除之前会询问是否进行该操作
rm -fr dir           #强制删除目录 dir 中的所有文件
```

8. ps 命令

该命令用于将某个时间点的进程运行情况选取下来并输出，**process** 之意，它的常用参数如下。

- **-A**: 所有的进程均显示出来。
- **-a**: 不与 **terminal** 有关的所有进程。
- **-u**: 有效用户的相关进程。
- **-x**: 一般与 **a** 参数一起使用，可列出较完整的信息。
- **-l**: 较长，较详细地将 **PID** 的信息列出。

其实我们只要记住 **ps** 一般使用的命令参数搭配即可，它们并不多，例如：

```
ps aux              #查看系统所有的进程数据
ps ax               #查看不与 terminal 有关的所有进程
ps -lA              #查看系统所有的进程数据
ps axjf             #查看连同一部分进程树状态
```

9. kill 命令

该命令用于向某个进程或者是某个 **PID**（一个数值）传送一个信号，它通常与 **ps** 和 **jobs** 命令一起使用，它的基本语法如下。

```
kill -signal PID
```

signal 的常用参数如下。

- **1**: **SIGHUP**，启动被终止的进程。
- **2**: **SIGINT**，相当于输入 **Ctrl+C**，中断一个程序的进行。

- 9: SIGKILL, 强制中断一个进程的进程。
- 15: SIGTERM, 以正常的结束进程方式来终止进程。
- 17: SIGSTOP, 相当于输入 Ctrl+Z, 暂停一个进程的进程。

注意, 最前面的数字为信号的代号, 使用时可以用代号代替相应的信号。

例如:

```
kill -9 2345          #强制中断 2345 号进程
kill -SIGTERM %1      #以正常结束进程方式来终止第一个后台工作进程
                      #可用 jobs 命令查看后台中的第一个工作进程
kill -SIGHUP PID      #重新改动进程 ID 为 PID 的进程, PID 可用 ps 命令通过管道
                      #命令加上 grep 命令进行筛选获得
```

10. file 命令

该命令用于判断接在 file 命令后的文件的基本数据, 因为在 Linux 下文件的类型并不是以后缀为分的, 所以这个命令对我们来说就很有用了, 它的用法非常简单, 基本语法如下。

```
file filename
```

例如:

```
file ./test
```

11. tar 命令

该命令用于对文件进行打包, 默认情况并不会压缩, 如果指定了相应的参数, 它还会调用相应的压缩程序 (如 gzip 和 bzip 等) 进行压缩和解压, 它的常用参数如下。

- -c: 新建打包文件。
- -t: 查看打包文件的内容含有哪些文件名。
- -x: 解打包或解压缩的功能, 可以搭配 -C (大写) 指定解压的目录, 注意 -c、-t、-x 不能同时出现在同一条命令中。
- -j: 通过 bzip2 的支持进行压缩/解压缩。
- -z: 通过 gzip 的支持进行压缩/解压缩。
- -v: 在压缩/解压缩过程中, 将正在处理的文件名显示出来。
- -f filename: filename 为要处理的文件。
- -C dir: 指定压缩/解压缩的目录 dir。

例如:

- 压缩: tar -jcv -f filename.tar.bz2, 要被处理的文件或目录名称。

- 查询: `tar -jtv -f filename.tar.bz2`。
- 解压: `tar -jxv -f filename.tar.bz2 -C`, 欲解压缩的目录。

注意, 文件名并不定要以后缀 `tar.bz2` 结尾, 这里主要是为了说明使用的压缩程序为 `bzip2`。

12. cat 命令

该命令用于查看文本文件的内容, 后接要查看的文件名, 通常可用管道与 `more` 和 `less` 一起使用, 从而可以一页页地查看数据。例如:

```
cat text | less #查看 text 文件中的内容
```

注意, 这条命令也可以使用 `less text` 来代替。

13. chgrp 命令

该命令用于改变文件所属用户组, 它的使用非常简单, 它的基本用法如下。

```
chgrp [-R] dirname/filename
```

- **-R**: 进行递归的持续对所有文件和子目录更改。

例如:

```
chgrp users -R ./dir #递归地把 dir 目录下中的所有文件和子目录下所有文件的用户组修改为 users
```

14. chmod 命令

该命令用于改变文件的权限, 一般的用法如下。

```
chmod [-R] xyz 文件或目录
```

- **-R**: 进行递归的持续更改, 即连同子目录下的所有文件都会更改。

同时, `chmod` 还可以使用 `u` (`user`)、`g` (`group`)、`o` (`other`)、`a` (`all`) 和 `+` (加入)、`-` (删除)、`=` (设置) 跟 `rw` 搭配来对文件的权限进行更改。

例如:

```
chmod 0755 file #把 file 的文件权限改变为-rxwr-xr-x
chmod g+w file #向 file 的文件权限中加入用户组可写权限
```

15. vim 命令

该命令主要用于文本编辑, 它接一个或多个文件名作为参数, 如果文件存在就打开, 如果文件不存在就以该文件名创建一个文件。`vim` 是一个非常好用的文本编辑器, 它里面有很多非常好用的命令, 可以查阅 `vim` 常用操作的详细说明。

16. gcc 命令

对于一个用 Linux 开发 C 语言程序的人来说, 这个命令就非常重要了, 它用于把 C 语言的源程序文件, 编译成可执行程序。由于 g++ 的很多参数跟它非常相似, 所以这里只介绍 gcc 的参数, 它的常用参数如下。

- -o: output 之意, 用于指定生成一个可执行文件的文件名。
- -c: 用于把源文件生成目标文件 (.o), 并阻止编译器创建一个完整的程序。
- -I: 增加编译时搜索头文件的路径。
- -L: 增加编译时搜索静态链接库的路径。
- -S: 把源文件生成汇编代码文件。
- -lm: 表示标准库的目录中名为 libm.a 的函数库。
- -lpthread: 连接 NPTL 实现的线程库。
- -std=: 用于指定把使用的 C 语言的版本。

例如:

```
gcc -o test test.c -lm -std=c99
#把源文件 test.c 按照 c99 标准编译成可执行程序 test
gcc -S test.c
#把源文件 test.c 转换为相应的汇编程序源文件 test.s
```

17. time 命令

该命令用于测算一个命令 (即程序) 的执行时间, 它的使用非常简单, 就像平时输入命令一样, 不过在命令的前面加入一个 time 即可。例如:

```
time ./process
time ps aux
```

在程序或命令运行结束后, 在最后输出了三个时间, 它们分别是:

- user: 用户 CPU 时间, 命令执行完成花费的用户 CPU 时间, 即命令在用户态中执行时间总和。
- system: 系统 CPU 时间, 命令执行完成花费的系统 CPU 时间, 即命令在核心态中执行时间总和。
- real: 实际时间, 从 command 命令行开始执行到运行终止的消逝时间。

注意, 用户 CPU 时间和系统 CPU 时间之和为 CPU 时间, 即命令占用 CPU 执行的时间总和。实际时间要大于 CPU 时间, 因为 Linux 是多任务操作系统, 往往在执行一条命

令时，系统还要处理其他任务。另一个需要注意的问题是即使每次执行相同命令，但所花费的时间也是不一样的，其花费时间是与系统运行相关的。

18. cal 命令

该命令用于显示日历。例如：

```
cal 2016           #显示 2016 年全年的日历
cal 10 2016        #显示 2016 年 10 月的日历
```

19. shutdown 命令

关机命令。例如：

```
shutdown -r +2      #两分钟后关机并重新启动
shutdown -h 12:30    #在 12:30 关机
shutdown -h now      #立即关机
```

实际上，Linux 有三个常用的关机命令，即 `shutdown`、`halt` 和 `poweroff`。

`shutdown` 以一种安全的方式关闭系统。所有登录用户都可以看到关机信息提示，并且 `login` 将被阻塞，所有进程都将接收到 `SIGTERM` 信号。这可以使 `vi` 等程序有时间将处于编辑状态的文件进行存储，邮件和新闻程序进程则可以将所有缓冲池内的数据进行适当的清除等。`shutdown` 通过通知 `init` 进程，要求它改换运行级别来实现。运行级别 0 用来关闭系统，运行级别 6 用来重启系统，运行级别 1 用来使系统进入执行系统管理任务状态，如果没有给出 `-h` 或 `-r` 标志时，这是 `shutdown` 命令的默认工作状态。

`halt` 是最简单的关机命令，其实 `halt` 就是调用 `shutdown -h`。`halt` 执行时会杀死应用进程，执行 `sync` 系统调用，文件系统写操作完成后就会停止内核。

`poweroff` 在关闭计算机操作系统之后，最后还会发送 `ACPI` 指令，通知电源，最后切断电源供应，当然路由器等嵌入系统不支持 `ACPI` 的。

3.5 本章小结

本章介绍了 Linux 操作系统基本知识，包括 Linux 系统的特点和组成，文件目录结构。重点介绍了在 Windows 操作系统上安装 VMware Workstation，以及在 VMware 平台上安装 Linux 集群的方法。我们详细描述了集群的配置方法，包括虚拟机主机名的修改方法、网络地址修改、网络连通性测试、Java 开发与运行环境 JDK 的安装方法，以及集群的免密钥登录配置。最后，我们建议读者多练习一些常见的 Linux 命令。

完成本章的学习和操作，将为后续的 Hadoop 安装、配置与应用打下基础。

HDFS 即 Hadoop 分布式文件系统 (Hadoop Distributed File System)，它的设计目标是把超大数据集存储到网络中的多台普通商用计算机上，并提供高可靠性和高吞吐率的服务。分布式文件系统要比普通磁盘文件系统复杂，因为它要引入网络编程；分布式文件系统要容忍节点失效，这也是一个很大的挑战。

本章主要介绍 HDFS 的架构、运行原理、安装配置和基本应用方法。

4.1 HDFS 概述

4.1.1 特点

HDFS 的设计前提和目标如下。

(1) 专为存储超大文件而设计：HDFS 应该能够支持 GB 级别大小的文件；它应该能够提供很大的数据带宽并且能够在集群中拓展到成百上千个节点；它的一个实例应该能够支持千万数量级别的文件。

(2) 适用于流式的数据访问：HDFS 适用于批处理的情况而不是交互式处理；它的重点是保证高吞吐量而不是低延迟的用户响应。

(3) 容错性：完善的冗余备份机制。

(4) 支持简单的一致性模型：HDFS 需要支持一次写入多次读取的模型，而且写入过程文件不会经常变化。

(5) 移动计算优于移动数据：HDFS 提供了使应用计算移动到离它最近数据位置的接口。

(6) 兼容各种硬件和软件平台。

HDFS 不适合的场景如下。

(1) 大量小文件：文件的元数据都存储在 NameNode 内存中，大量小文件意味着元数据的增加，会占用大量内存。

(2) 低延迟数据访问：HDFS 是专门针对高数据吞吐量而设计的。

(3) 多用户写入：因为会导致一致性维护的困难。

4.1.2 主要组件与架构

HDFS 主要由 3 个组件构成，分别是 NameNode、SecondaryNameNode 和 DataNode，HDFS 是以 Master/Slave（主从）模式运行的，其中 NameNode、SecondaryNameNode 运行在 Master 节点，DataNode 运行 Slave 节点上。

NameNode 和 DataNode 架构如图 4-1 所示。

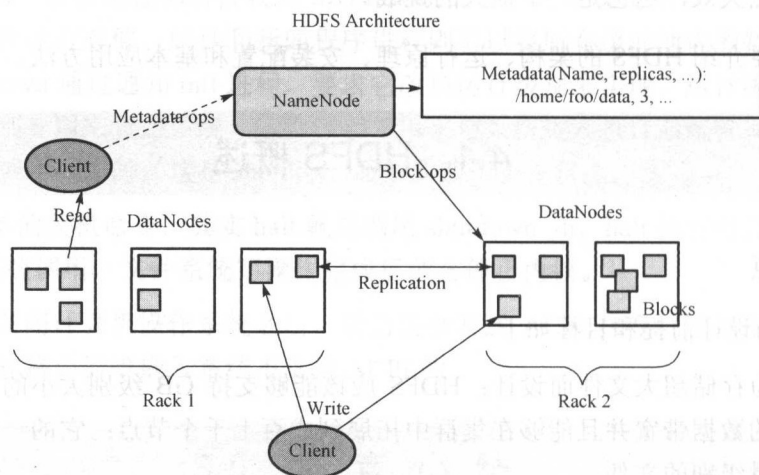


图 4-1 NameNode 和 DataNode 架构

4.2 HDFS 架构分析

4.2.1 数据块

磁盘数据块是磁盘读写的基本单位，与普通文件系统类似，HDFS 也会把文件分块来

存储。HDFS 默认数据块大小为 64 MB，而磁盘块一般为 512 B。HDFS 块为何如此之大呢？块增大可以减少寻址时间与文件传输时间的比例，若寻址时间为 10 ms，磁盘传输速率为 100 MB/s，那么寻址与传输比仅为 1%。当然，磁盘块太大也不好，因为一个 MapReduce 通常以一个块作为输入，块过大会导致整体任务数量过小，降低作业处理速度。

数据块是存储在 **DataNode** 中的，为了能够容错，数据块是以多个副本的形式分布在集群中的，副本数量默认为 3，后面会专门介绍数据块的复制机制。

HDFS 按块存储还有如下好处。

(1) 文件可以任意大，也不用担心单个节点磁盘容量小于文件的情况。

(2) 简化了文件子系统的设计，子系统只存储文件块数据，而文件元数据则交由其他系统 (**NameNode**) 管理。

(3) 有利于备份和提高系统可用性，这得益于以块为单位进行备份的设计，HDFS 默认备份数量为 3。

(4) 有利于负载均衡。

4.2.2 NameNode

1. NameNode 中的元信息

当一个客户端请求一个文件或者存储一个文件时，它需要先知道具体到哪个 **DataNode** 上存取，获得这些信息后，客户端再直接和这个 **DataNode** 进行交互，而这些信息的维护者就是 **NameNode**。

NameNode 管理着文件系统的命名空间，它维护文件系统树及树中的所有文件和目录。**NameNode** 也负责维护所有这些文件或目录的打开、关闭、移动、重命名等操作。对于实际文件数据的保存与操作，都是由 **DataNode** 负责的。当一个客户端请求数据时，它仅仅是从 **NameNode** 中获取文件的元信息，而具体的数据传输不需要经过 **NameNode**，是由客户端直接与相应的 **DataNode** 进行交互的。

NameNode 保存元信息的种类有：

- 文件名目录名及它们之间的层级关系；
- 文件目录的所有者及其权限；
- 每个文件块的名及文件有哪些块组成。

需要注意的是，**NameNode** 元信息并不包含每个块的位置信息，这些信息会在

NameNode 启动时从各个 DataNode 获取并保存在内存中，因为这些信息会在系统启动时由数据节点重建。把块位置信息放在内存中，在读取数据时会减少查询时间，增加读取效率。NameNode 也会实时通过心跳机制和 DataNode 进行交互，实时检查文件系统是否运行正常。不过 NameNode 元信息会保存各个块的名称及文件由哪些块组成。

一般来说，一条元信息记录会占用 200 B 内存空间。假设块大小为 64 MB，备份数量是 3，那么一个 1 GB 大小的文件将占用 $16 \times 3 = 48$ 个文件块。如果现在有 1000 个 1 MB 大小的文件，则会占用 $1000 \times 3 = 3000$ 个文件块（多个文件不能放到一个块中）。我们可以发现，如果文件越小，存储同等大小文件所需要的元信息就越多，所以，Hadoop 更喜欢大文件。

2. 元信息的持久化

在 NameNode 中存放元信息的文件是 fsimage。在系统运行期间所有对元信息的操作都保存在内存中并被持久化到另一个文件 edits 中，并且 edits 文件和 fsimage 文件会被 SecondaryNameNode 周期性地合并（合并过程会在 SecondaryNameNode 中详细介绍）。

运行 NameNode 会占用大量内存和 I/O 资源，一般 NameNode 不会存储用户数据或执行 MapReduce 任务。

为了简化系统的设计，Hadoop 只有一个 NameNode，这也就导致了 Hadoop 集群的单点故障问题。因此，对 NameNode 节点的容错尤其重要，Hadoop 提供了如下两种机制来解决。

（1）将 Hadoop 元数据写入到本地文件系统的同时再实时同步到一个远程挂载的网络文件系统（NFS）。

（2）运行一个 SecondaryNameNode，它的作用是与 NameNode 进行交互，定期通过编辑日志文件合并命名空间镜像。当 NameNode 发生故障时，它会通过自己合并的命名空间镜像副本来恢复。需要注意的是 SecondaryNameNode 保存的状态总是滞后于 NameNode，所以这种方式难免会导致丢失部分数据。

4.2.3 DataNode

DataNode 是 HDFS 中的 Worker 节点，它负责存储数据块，也负责为系统客户端提供数据块的读写服务，同时还会根据 NameNode 的指示来进行创建、删除和复制等操作。此外，它还会通过心跳定期向 NameNode 发送所存储文件块列表信息。当对 HDFS 文件系统进行读写时，NameNode 告知客户端每个数据驻留在哪个 DataNode，客户端直接与 DataNode 进行通信，DataNode 还会与其他 DataNode 通信，复制这些块以实现冗余。

4.2.4 SecondaryNameNode

需要注意, SecondaryNameNode 并不是 NameNode 的备份。我们从前面的介绍已经知道, 所有 HDFS 文件的元信息都保存在 NameNode 的内存中。在 NameNode 启动时, 它首先会加载 fsimage 到内存中, 在系统运行期间, 所有对 NameNode 的操作也都保存在内存中, 同时为了防止数据丢失, 这些操作又会不断被持久化到本地 edits 文件中。

edits 文件的目的是为了提高系统的操作效率, NameNode 在更新内存中的元信息之前都会先将操作写入 edits 文件。在 NameNode 重启的过程中, edits 会和 fsimage 合并到一起, 但是合并的过程会影响到 Hadoop 重启的速度, SecondaryNameNode 就是为了解决这个问题而诞生的。

SecondaryNameNode 的角色就是定期合并 edits 和 fsimage 文件, 我们来看一下合并的步骤。

(1) 合并之前告知 NameNode 把所有的操作写到新的 edits 文件并将其命名为 edits.new。

(2) SecondaryNameNode 从 NameNode 请求 fsimage 和 edits 文件。

(3) SecondaryNameNode 把 fsimage 和 edits 文件合并成新的 fsimage 文件。

(4) NameNode 从 SecondaryNameNode 获取合并好的新的 fsimage 并将旧的替换掉, 并把 edits 用 (1) 创建的 edits.new 文件替换掉。

(5) 更新 fstime 文件中的检查点。

最后再总结一下整个过程中涉及 NameNode 中的相关文件。

- fsimage: 保存的是上个检查点的 HDFS 的元信息。
- edits: 保存的是从上个检查点开始发生的 HDFS 元信息状态改变信息。
- fstime: 保存了最后一个检查点的时间戳。

4.2.5 数据备份

HDFS 通过备份数据块的形式来实现容错, 除了文件的最后一个数据块外, 其他所有数据块大小都是一样的。数据块的大小和备份因子都是可以配置的。NameNode 负责各个数据块的备份, DataNode 会通过心跳的方式定期向 NameNode 发送自己节点上的 Block 报告, 这个报告中包含了 DataNode 节点上的所有数据块的列表。

文件副本的分布位置直接影响着 HDFS 的可靠性和性能。一个大型的 HDFS 文件系

统一一般都是需要跨很多机架的，不同机架之间的数据传输需要经过网关，并且同一个机架中机器之间的带宽要大于不同机架机器之间的带宽。如果把所有的副本都放在不同的机架中，这样既可以防止机架失败导致数据块不可用，又可以在读数据时利用到多个机架的带宽，并且也可以很容易实现负载均衡。但是，如果是写数据，各个数据块需要同步到不同的机架，会影响到写数据的效率。

而在 Hadoop 中，如果在副本数量为 3 的情况下（Hadoop 默认是这么存放的），会把第一个副本放到机架的一个节点上，第二个副本放到同一个机架的另一个节点上，把最后一个节点放到不同的机架上。这种策略减少了跨机架副本的个数，提高了写的性能，也能够允许一个机架失败的情况，算是一个很好的权衡。

关于副本的选择，在读的过程中，HDFS 会选择最近的一个副本给请求者。

关于安全模式，当 Hadoop 的 NameNode 节点启动时，会进入安全模式阶段。在此阶段，DataNode 会向 NameNode 上传它们数据块的列表，让 NameNode 得到块的位置信息，并对每个文件对应的数据块副本进行统计。当最小副本条件满足时，即一定比例的数据块都达到最小副本数，系统就会退出安全模式，而这需要一定的延迟时间。当最小副本条件未达到要求时，就会对副本数不足的数据块安排 DataNode 进行复制，直至达到最小副本数。而在安全模式下，系统会处于只读状态，NameNode 不会处理任何块的复制和删除命令。

4.2.6 通信协议

所有的 HDFS 中的沟通协议都是基于 TCP/IP 协议的，一个客户端通过指定的 TCP 端口与 NameNode 机器建立连接，并通过 Client Protocol 协议与 NameNode 交互。而 DataNode 则通过 DataNode Protocol 协议与 NameNode 进行沟通。HDFS 的 RCP（远程过程调用）对 Client Protocol 和 DataNode Protocol 做了封装。按照 HDFS 的设计，NameNode 不会主动发起任何请求，只会被动接收来自客户端或 DataNode 的请求。

4.2.7 可靠性保证

HDFS 可以允许 DataNode 失败。DataNode 会定期（默认为 3 s）向 NameNode 发送心跳，若 NameNode 在指定时间间隔内没有收到心跳，它就认为此节点已经失败。此时，NameNode 把失败节点的数据（从另外的副本节点获取）备份到另外一个健康的节点，这就保证了集群始终维持指定的副本数。

HDFS 可以检测到数据块损坏。在读取数据块时，HDFS 会对数据块和保存的校验和文件匹配，如果发现不匹配，NameNode 同样会重新备份损坏的数据块。

4.3 文件操作过程分析

4.3.1 读文件

HDFS 有一个文件系统实例，客户端通过调用这个实例的 `open()` 方法就可以打开系统中希望读取的文件。HDFS 通过 RPC 调用 `NameNode` 获取文件块的位置信息，对于文件的每一个块，`NameNode` 会返回含有该块副本的 `DataNode` 的节点地址。另外，客户端还会根据网络拓扑来确定它与每一个 `DataNode` 的位置信息，从离它最近的那个 `DataNode` 获取数据块的副本，最理想的情况是数据块就存储在客户端所在的节点上。

HDFS 会返回一个 `FSDaataInputStream` 对象，`FSDaataInputStream` 类转而封装成 `DFSDataInputStream` 对象，这个对象管理着与 `DataNode` 和 `NameNode` 的 I/O，具体过程是：

- (1) 客户端发起读请求。
- (2) 客户端与 `NameNode` 得到文件的块及位置信息列表。
- (3) 客户端直接和 `DataNode` 交互读取数据。
- (4) 读取完成关闭连接。

图 4-2 给出了上述读文件的过程示意。

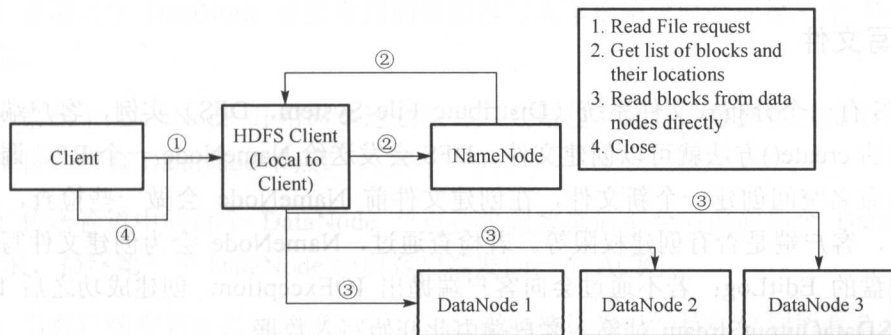


图 4-2 HDFS 读文件过程

当 `FSDaataInputStream` 与 `DataNode` 通信时遇到错误，它会选取另一个较近的 `DataNode`，并为出故障的 `DataNode` 做标记以免重复向其读取数据。`FSDaataInputStream` 还会对读取的数据块进行校验和确认，发现块损坏时也会重新读取并通知 `NameNode`。

这样设计的巧妙之处有：

(1) 让客户端直接联系 **DataNode** 检索数据，可以使 **HDFS** 扩展到大量的并发客户端，因为数据流就是分散在集群的每个节点上的，在运行 **MapReduce** 任务时，每个客户端就是一个 **DataNode** 节点。

(2) **NameNode** 仅需要相应块的位置信息请求（位置信息在内存中，速度极快），否则随着客户端的增加，**NameNode** 会很快成为瓶颈。

这里有必要理解 **Hadoop** 的网络拓扑。在海量数据处理过程中，主要限制因素是节点之间的带宽。衡量两个节点之间的带宽往往很难实现，在这里 **Hadoop** 采取了一个简单的方法，它把网络拓扑看成一棵树，两个节点的距离等于它们到最近共同祖先距离的总和，而树的层次可以这么划分：

- 同一节点中的进程；
- 同一机架上的不同节点；
- 同一数据中心不同机架；
- 不同数据中心的节点。

例如，若数据中心 **d1** 中一个机架 **r1** 中一个节点 **n1** 表示为 **d1/r1/n1**，则：

```
distance(d1/r1/n1,d1/r1/n1)=0;
distance(d1/r1/n1,d1/r1/n2)=2;
distance(d1/r1/n1,d1/r2/n3)=4;
distance(d1/r1/n1,d2/r3/n4)=6;
```

4.3.2 写文件

HDFS 有一个分布式文件系统（**Distribute File System, DFS**）实例，客户端通过调用这个实例的 **create()** 方法就可以创建文件。**DFS** 会发送给 **NameNode** 一个 **RPC** 调用，在文件系统的命名空间创建一个新文件，在创建文件前 **NameNode** 会做一些检查，看看文件是否存在，客户端是否有创建权限等。若检查通过，**NameNode** 会为创建文件写一条记录到本地磁盘的 **EditLog**；若不通过会向客户端抛出 **IOException**。创建成功之后 **DFS** 会返回一个 **FSDDataOutputStream** 对象，客户端由此开始写入数据。

同读文件过程一样，**FSDDataOutputStream** 类转而封装成 **DFSDataOutputStream** 对象，这个对象管理着与 **DataNode** 和 **NameNode** 的 **I/O**，具体过程是：

- (1) 客户端在向 **NameNode** 请求之前先写入文件数据到本地文件系统的一个临时文件。
- (2) 待临时文件达到块大小时开始向 **NameNode** 请求 **DataNode** 信息。

(3) NameNode 在文件系统中创建文件并返回给客户端一个数据块及其对应 DataNode 的地址列表（列表中包含副本存放的地址）。

(4) 客户端通过上一步得到的信息把创建临时文件块 Flush 到列表中的第一个 DataNode。

(5) 当文件关闭，NameNode 会提交这次文件创建，此时文件在文件系统中可见。

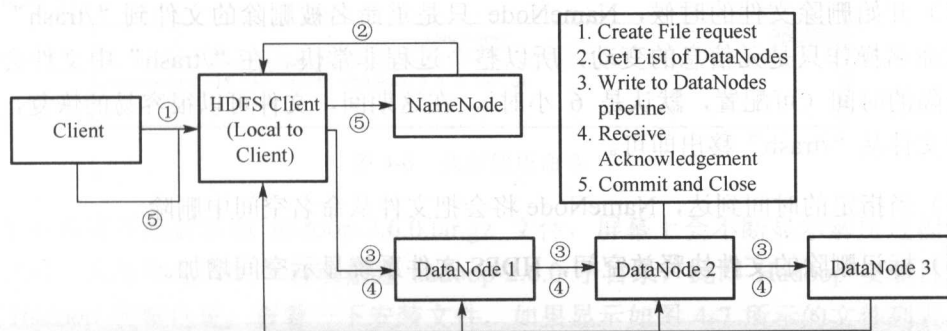


图 4-3 HDFS 写文件过程

上面 (4) 描述的 Flush 过程实际处理过程比较复杂，现在单独描述一下。

(1) 首先，第一个 DataNode 是以数据包（数据包一般 4 KB）的形式从客户端接收数据的，DataNode 在把数据包写入到本地磁盘的同时会向第二个 DataNode（作为副本节点）传送数据。

(2) 在第二个 DataNode 把接收到的数据包写入本地磁盘时会向第三个 DataNode 发送数据包。

(3) 第三个 DataNode 开始向本地磁盘写入数据包。此时，数据包以流水线的形式被写入和备份到所有 DataNode 节点。

(4) 传送管道中的每个 DataNode 节点在收到数据后都会向前面那个 DataNode 发送一个 ACK，最终第一个 DataNode 会向客户端发回一个 ACK。

(5) 当客户端收到数据块的确认之后，数据块被认为已经持久化到所有节点，然后客户端会向 NameNode 发送一个确认。

(6) 如果管道中的任何一个 DataNode 失败，管道会被关闭，数据将会继续写到剩余的 DataNode 中。同时 NameNode 会被告知待备份状态，NameNode 会继续备份数据到新的可用的节点。

(7) 数据块都会通过计算校验和来检测数据的完整性, 校验和以隐藏文件的形式被单独存放在 HDFS 中, 供读取时进行完整性校验。

4.3.3 删除文件

HDFS 文件删除过程一般需要如下几步。

(1) 开始删除文件的时候, NameNode 只是重命名被删除的文件到 “/trash” 目录, 因为重命名操作只是元信息的变动, 所以整个过程非常快。在 “/trash” 中文件会被保留一定间隔的时间 (可配置, 默认是 6 小时), 在这期间, 文件可以很容易的恢复, 恢复只需要将文件从 “/trash” 移出即可。

(2) 当指定的时间到达, NameNode 将会把文件从命名空间中删除。

(3) 标记删除的文件块释放空间, HDFS 文件系统显示空间增加。

4.4 Hadoop HDFS 安装与配置

现在让我们开始安装 HDFS。

注意, 每一个节点的安装和配置是相同的。实际工作中, 通常在 Master 节点上完成安装和配置后, 然后将安装目录复制到其他节点即可。请注意, 这里的所有操作都是一般用户权限 (如 csu)。

4.4.1 解压 Hadoop 安装包

请读者在第 4 章软件资源目录下, 找到 Hadoop-2.6.0.tar.gz 文件, 请将其复制到 Master 节点的 “/home/csu” 下的 Resources 子目录内 (可以采用直接拖拽方式), 如图 4-4 所示。

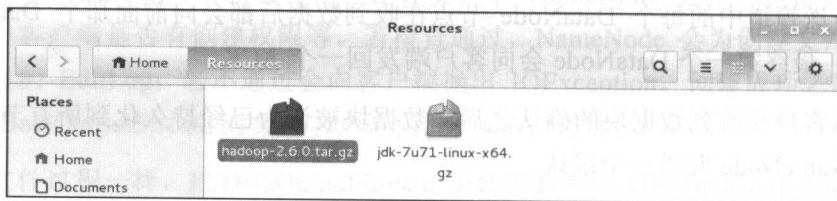


图 4-4 /home/csu/Resources 内的 Hadoop 压缩包

首先将 hadoop-2.6.0.tar.gz 文件复制到安装目录的上一级目录, 我们这里指定为 “/home/csu”, 所以, 执行 “cp /home/csu/Resources/hadoop-2.6.0.tar.gz ~/”, 如图 4-5 所示。

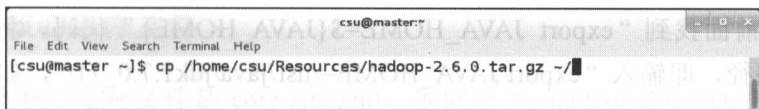


图 4-5 将 Hadoop 压缩包复制到/home/csu 下

进入 csu 目录，执行解压缩命令“tar -zxvf ~/hadoop-2.6.0.tar.gz”，即可实现安装，如图 4-6 所示。

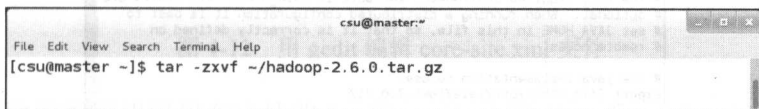


图 4-6 执行解压命令

回车后系统开始解压缩 hadoop-2.6.0.tar.gz 文件，屏幕上会不断显示解压过程信息，执行成功后，系统将在 csu 下自动创建 hadoop-2.6.0 子目录，此即 Hadoop 安装目录。我们进入 Hadoop 安装目录，查看一下安装文件，如果显示如图 4-7 所示的文件列表，说明解压缩成功。

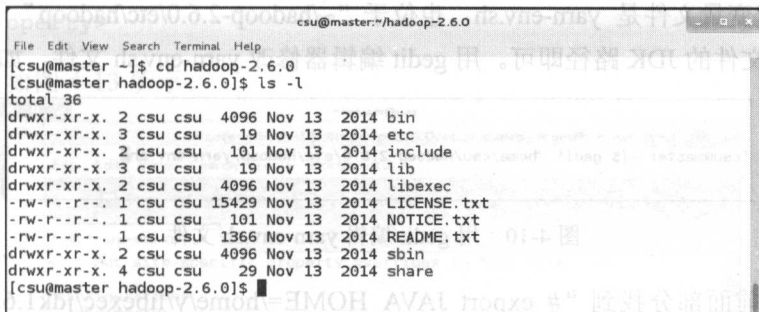


图 4-7 进入 hadoop-2.6.0 目录查看安装文件

要使用 Hadoop，还需要完成一系列配置。

4.4.2 配置 Hadoop 环境变量

Hadoop 环境变量文件是 hadoop-env.sh，它位于“~/hadoop-2.6.0/etc/hadoop”子目录下。我们只需要配置该文件的 JDK 路径即可。用 gedit 编辑器修改 hadoop-env.sh 文件，如图 4-8 所示。

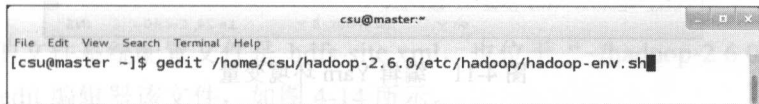


图 4-8 用 gedit 编辑 hadoop-env.sh 文件

在文件的前面找到“`export JAVA_HOME=${JAVA_HOME}`”代码，将其修改为实际的 JDK 安装路径，即输入“`export JAVA_HOME=/usr/java/jdk1.7.0_71/`”，如图 4-9 所示。

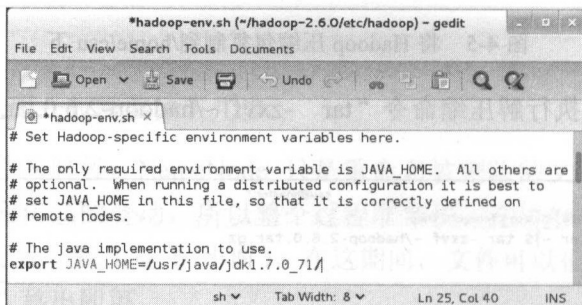


图 4-9 编辑 Hadoop 环境变量

编辑完毕，保存退出即可。

4.4.3 配置 Yarn 环境变量

Yarn 环境变量文件是 `yarn-env.sh`，也位于“`~/hadoop-2.6.0/etc/hadoop`”子目录下。我们只需要配置该文件的 JDK 路径即可。用 gedit 编辑器修改 `yarn-env.sh` 文件，如图 4-10 所示。

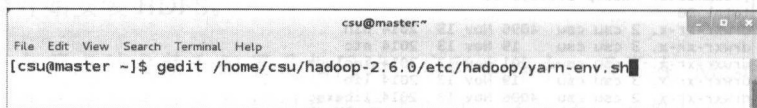


图 4-10 用 gedit 编辑 yarn-env.sh 文件

在文件的前面部分找到“`# export JAVA_HOME=/home/y/libexec/jdk1.6.0/`”，将其修改为“`export JAVA_HOME=/usr/java/jdk1.7.0_71/`”，如图 4-11 所示。

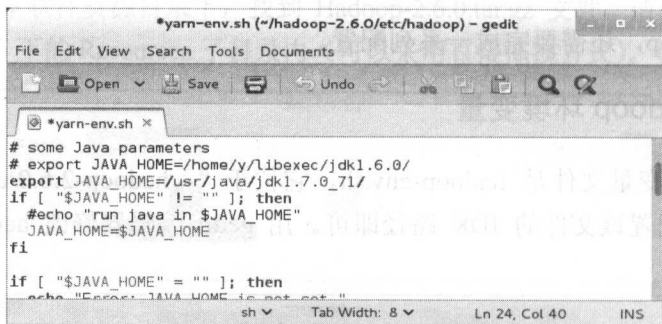


图 4-11 编辑 Yarn 环境变量

编辑完毕，保存退出即可。

4.4.4 配置核心组件文件

Hadoop 的核心组件文件是 `core-site.xml`，也位于“`~/hadoop-2.6.0/etc/hadoop`”子目录下。用 `gedit` 编辑器 `core-site.xml` 文件，如图 4-12 所示。

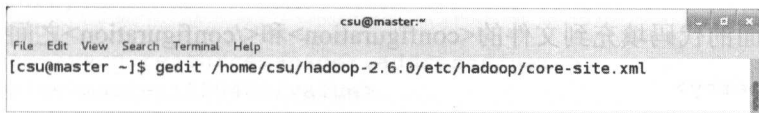


图 4-12 用 `gedit` 编辑 `core-site.xml` 文件

需要将下面的配置代码放在文件的 `<configuration>` 和 `</configuration>` 之间。

```
<property>
  <name>fs.defaultFS</name>
  <value>hdfs://master:9000</value>
</property>
<property>
  <name>hadoop.tmp.dir</name>
  <value>/home/csu/hadoopdata</value>
</property>
```

输入结果如图 4-13 所示。

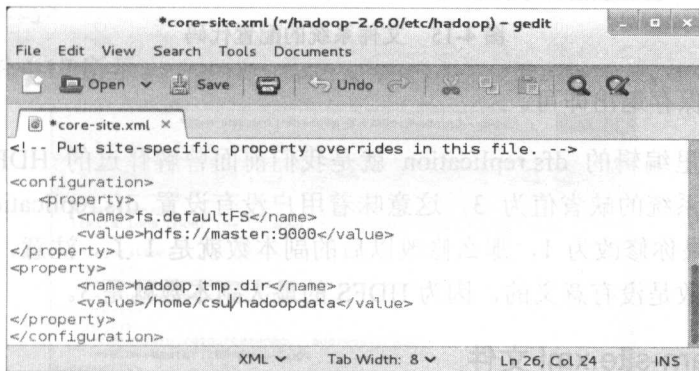


图 4-13 编辑核心组件配置代码

编辑完毕，保存退出即可。

4.4.5 配置文件系统

Hadoop 的文件系统配置文件是 `hdfs-site.xml`，也位于“`~/hadoop-2.6.0/etc/hadoop`”子目录下。用 `gedit` 编辑器该文件，如图 4-14 所示。

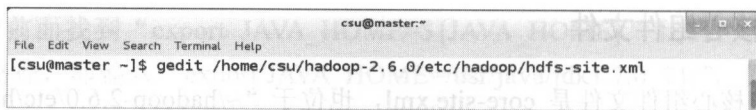


图 4-14 用 gedit 编辑 hdfs-site.xml 文件

需要将下面的代码填充到文件的<configuration>和</configuration>之间。

```
<property>
  <name>dfs.replication</name>
  <value>1</value>
</property>
```

输入结果如图 4-15 所示。

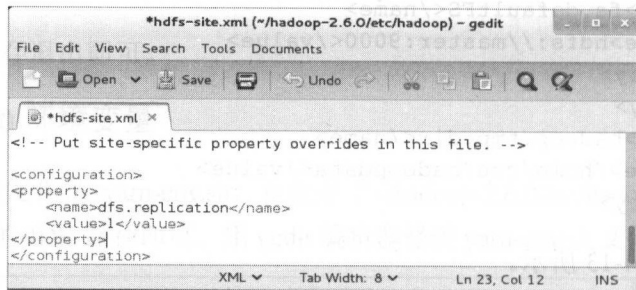


图 4-15 文件系统的配置代码

编辑完毕，保存退出即可。

实际上，这里编辑的 dfs.replication 就是我们前面曾解释过的 HDFS 数据块的副本数。我们知道，系统的缺省值为 3，这意味着用户没有设置 dfs.replication 的时候，副本数是 3。但是如果你修改为 1，那么修改以后的副本数就是 1 了。注意，把 dfs.replication 配置成超过 3 的数是没有意义的，因为 HDFS 的最大副本数就是 3。

4.4.6 配置 yarn-site.xml 文件

Yarn 的站点配置文件是 yarn-site.xml，也位于“~/hadoop-2.6.0/etc/hadoop”子目录下。用 gedit 编辑器该文件，如图 4-16 所示。

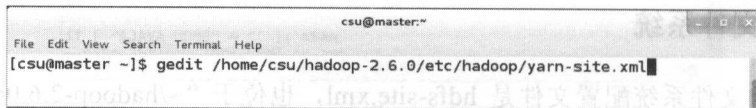


图 4-16 用 gedit 编辑 yarn-site.xml 文件

需要将下面的代码填充到文件的<configuration>和</configuration>之间。

```
<property>
  <name>yarn.nodemanager.aux-services</name>
  <value>mapreduce_shuffle</value>
</property>
<property>
  <name>yarn.resourcemanager.address</name>
  <value>master:18040</value>
</property>
<property>
  <name>yarn.resourcemanager.scheduler.address</name>
  <value>master:18030</value>
</property>
<property>
  <name>yarn.resourcemanager.resource-tracker.address</name>
  <value>master:18025</value>
</property>
<property>
  <name>yarn.resourcemanager.admin.address</name>
  <value>master:18141</value>
</property>
<property>
  <name>yarn.resourcemanager.webapp.address</name>
  <value>master:18088</value>
</property>
```

输入结果如图 4-17 所示。

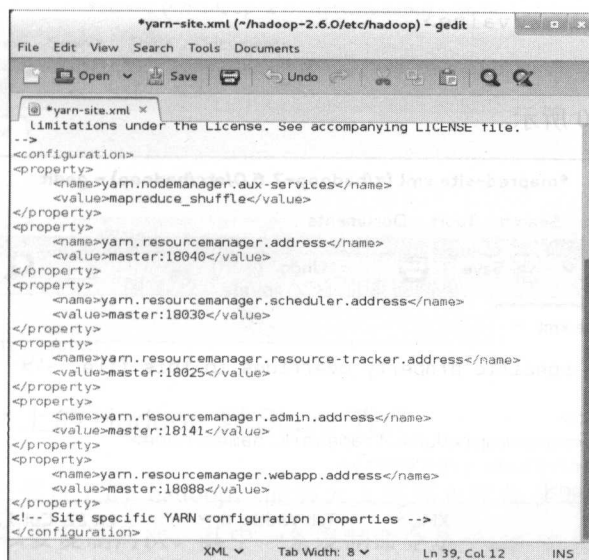


图 4-17 yarn-site.xml 文件的配置代码

编辑完毕，保存退出即可。

4.4.7 配置 MapReduce 计算框架文件

在~/hadoop-2.6.0/etc/hadoop 子目录下，系统已经有一个 mapred-site.xml.template 文件，我们需要将其复制并改名，位置不变，命令是“cp ~/hadoop-2.6.0/etc/hadoop/mapred-site.xml.template ~/~/hadoop-2.6.0/etc/hadoop/mapred-site.xml”，如图 4-18 所示。

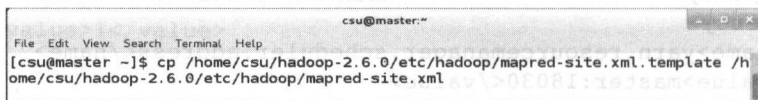


图 4-18 复制并改名 mapred-site.xml.template 文件

然后，用 gedit 编辑 mapred-site.xml 文件，如图 4-19 所示。

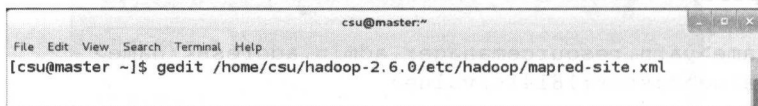


图 4-19 用 gedit 编辑 mapred-site.xml 文件

需要将下面的代码填充到文件的<configuration>和</configuration>之间。

```
<property>
  <name>mapreduce.framework.name</name>
  <value>yarn</value>
</property>
```

输入结果如图 4-20 所示。

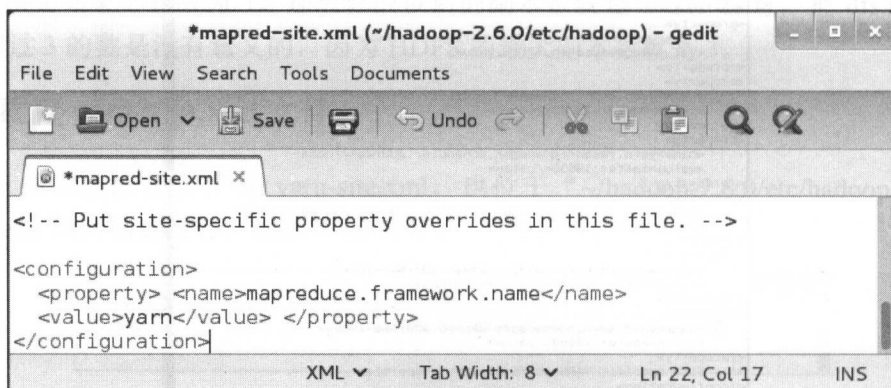


图 4-20 mapred-site.xml 文件的配置代码

编辑完毕，保存退出即可。

4.4.8 配置 Master 的 slaves 文件

slaves 文件给出了 Hadoop 集群的 Slave 节点列表。该文件十分重要，因为启动 Hadoop 的时候，系统总是根据当前 slaves 文件中 Slave 节点名称列表启动集群，不在列表中的 Slave 节点便不会被视为计算节点。

用 gedit 编辑 slaves 文件，如图 4-21 所示。

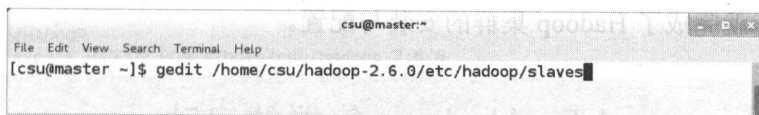


图 4-21 用 gedit 编辑 slaves 文件

读者应当根据自己所搭建集群的实际情况进行编辑。例如，我们这里由于已经安装了 Slave0 和 Slave1，并且计划将它们全部投入 Hadoop 集群运行，所以应当输入如下代码。

```
slave0
slave1
```

注意，文件中原来有缺省的 localhost，应将其删除。输入结果如图 4-22 所示。

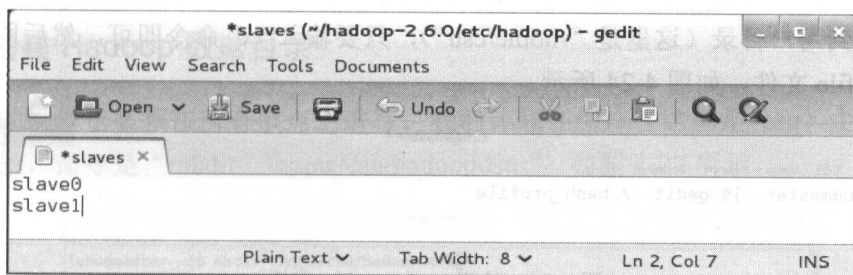


图 4-22 slaves 文件的编辑结果

编辑完毕，保存退出即可。

4.4.9 复制 Master 上的 Hadoop 到 Slave 节点

通过复制 Master 节点上的 hadoop，能够大大提高系统部署效率。由于我们这里有 Slave0 和 Slave1，所以要复制两次。其中一条复制命令是“`scp -r /home/csu/hadoop-2.6.0 csu@slave0:~/`”，如图 4-23 所示。

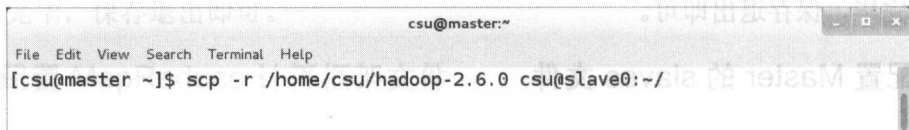


图 4-23 复制 Master 节点上的 Hadoop 到 Slave0 节点

注意，由于我们前面已经配置了免密钥登录，因此这里不会有密码输入认证，回车后立即开始复制。

至此，我们就完成了 Hadoop 集群的安装与配置。

4.5 Hadoop 集群的启动

现在是启动 Hadoop 集群的时候了，但是，首次启动 Hadoop 还需要做一些准备工作。

4.5.1 配置操作系统环境变量

由于我们是在 Linux 集群上安装的 Hadoop 集群，所以自然需要配置 Linux 系统平台的环境变量。注意，这里的配置需要在集群的所有计算机上进行，并且使用一般用户权限。下面演示在 Master 节点上的配置。

首先回到用户目录（这里是“/home/csu”），只要输入 `cd` 命令即可，然后用 `gedit` 编辑 `.bash_profile` 文件，如图 4-24 所示。

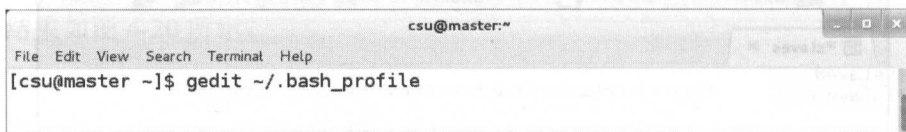


图 4-24 编辑 .bash_profile 文件

将下述代码追加到文件的尾部：

```
#HADOOP
export HADOOP_HOME=/home/csu/hadoop-2.6.0
export PATH=$HADOOP_HOME/bin:$HADOOP_HOME/sbin:$PATH
```

输入结果如图 4-25 所示。实际上我们在第 2 章安装 JDK 时候曾经配置过该文件，这里需要类似地配置 Hadoop 安装路径。

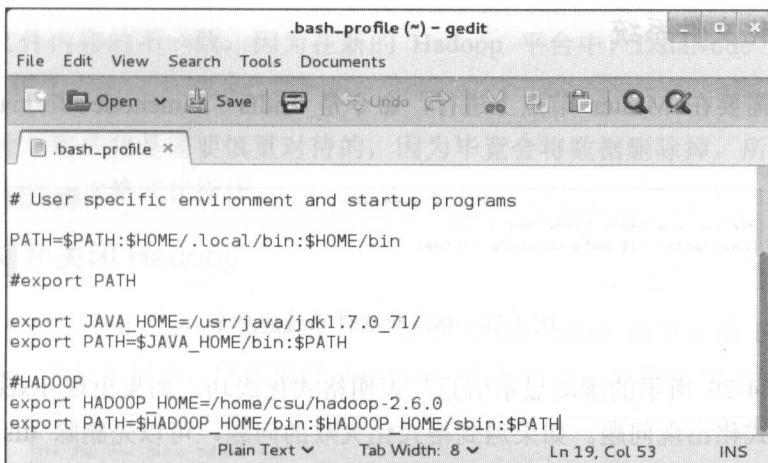


图 4-25 输入的 Hadoop 环境变量

保存退出后, 执行“source ~/.bash_profile”命令, 使上述配置生效, 如图 4-26 所示。

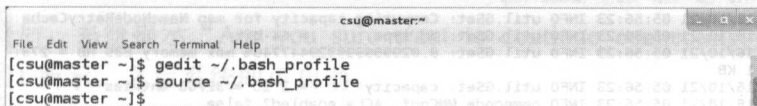


图 4-26 使.bash_profile 文件配置生效

请读者注意, 也要在其他节点进行上述配置。

4.5.2 创建 Hadoop 数据目录

本节的操作也必须在所有节点上进行。我们计划在用户（这里是 csu）主目录下, 创建数据目录, 命令是“mkdir /home/csu/hadoopdata”, 如图 4-27 所示。

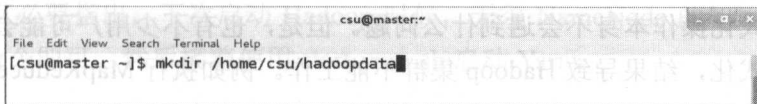


图 4-27 创建数据目录 hadoopdata

请读者注意, 这里的数据目录名“hadoopdata”与前面 Hadoop 核心组件文件 core-site.xml 中的配置:

```

<name>hadoop.tmp.dir</name>
<value>/home/csu/hadoopdata</value>

```

是一致的。

4.5.3 格式化文件系统

该操作只需要在 Master 节点上进行，命令是“`hdfs namenode -format`”，如图 4-28 所示。

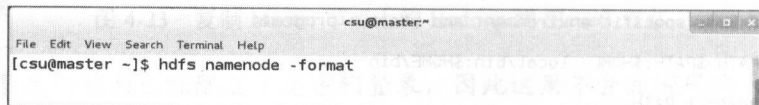


图 4-28 格式化文件系统的命令

看到如图 4-29 所示的滚动显示信息，表明格式化成功。如果出现 Exception/Error 信息，则表示格式化出现问题。如果遇到格式化失败的问题，可以先删除 `dfs.name.dir` 参数指定的目录，确保该目录不存在，然后实施格式化。Hadoop 这样做的目的是防止错误地将已存在的集群格式化了。

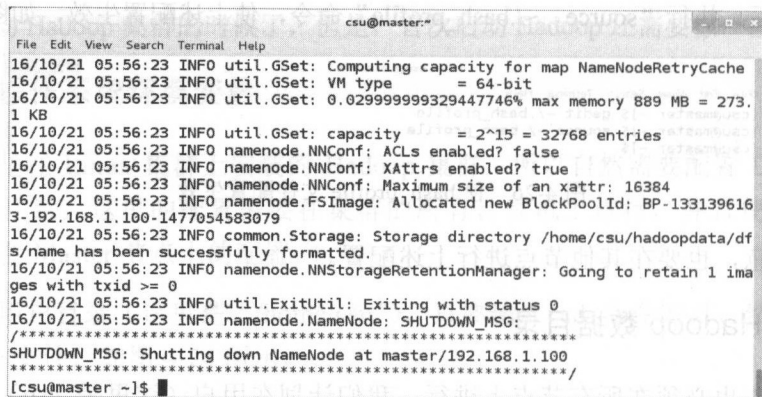


图 4-29 格式化文件系统的命令

通常，格式化操作本身不会遇到什么问题。但是，也有不少用户可能会由于种种原因而多次进行格式化，结果导致 Hadoop 集群不能工作。例如执行 MapReduce 程序会遇到类似“`There are 0 datanode(s) running and no node(s) are excluded in this operation.`”这样的异常信息。

实际上，这样的状况在早期的 Hadoop 版本中是存在的，主要原因是多次格式化后，导致 NameNode 中 `Hadoopdata/dfs/name/current/` 下文件 `VERSION` 的内容与 DataNode 中的同名文件内容不一致（具体是 `ClusterID` 不一致），解决办法就是通过人工编辑使两个文件一致即可，或者强制删除这些文件再重新格式化（当然要先关闭 Hadoop）。但是，2.6 版本以后的 Hadoop 已经不需要这样做了，用户可以多次进行格式化，并不会导致

VERSION 文件内容的不一致，因为在新的 Hadoop 平台中，DataNode 上已经没有这个 VERSION 文件了。

但是，重新格式化是需要慎重对待的，因为毕竟会将数据删除掉，所以，有一些维护人员建议只进行一次格式化操作。

4.5.4 启动和关闭 Hadoop

完成上述准备即可启动 Hadoop 了。可以使用 start-all.sh 命令启动 Hadoop 集群。首先进入 Hadoop 安装主目录，然后执行 sbin/start-all.sh 命令，如图 4-30 所示。

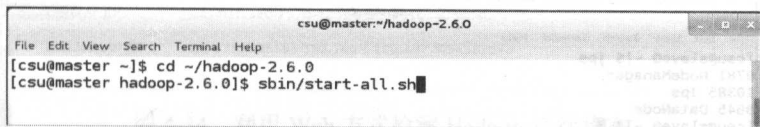


图 4-30 启动 Hadoop 集群的命令

执行命令后，系统提示 “Are you sure want to continue connecting(yes/no)”，请输入 yes，如图 4-31 所示，之后系统即可启动。

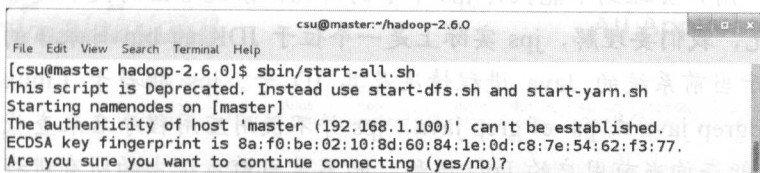


图 4-31 启动中的人机交互

要关闭 Hadoop 集群，可以使用 stop-all.sh 命令。

但是，有必要指出，下次启动 Hadoop 时候，无须 NameNode 的初始化，只需要使用 start-dfs.sh 命令即可，然后接着使用 start-yarn.sh 启动 Yarn。

实际上，Hadoop 系统建议放弃 (deprecated) 使用 start-all.sh 和 stop-all.sh 一类的命令，而改用 start-dfs.sh 和 start-yarn.sh 命令。

4.5.5 验证 Hadoop 是否启动成功

用户可以在终端执行 jps 命令查看 Hadoop 是否启动成功。在 Master 节点，执行 jps 后如果显示的结果是四个进程的名称：SecondaryNameNode、ResourceManager、Jps 和 NameNode，如图 4-32 所示，则表明主节点 (Master) 启动成功。

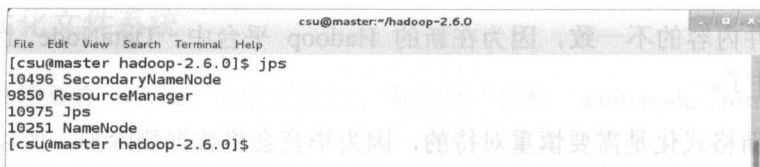


图 4-32 用 jps 命令查看主节点中的进程

在 Slave0 节点执行 jps 命令，打印的结果中会显示三个进程，分别是 NodeManager、Jps 和 DataNode，如图 4-33 所示，表明从节点（Slave0）启动成功。其他节点可以类似验证。

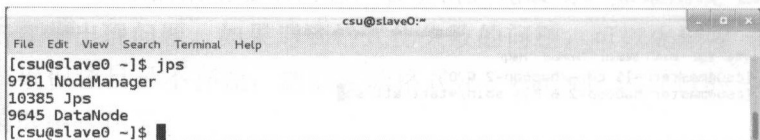


图 4-33 用 jps 命令查看从节点中的进程

注意事项

有时候，用户会碰到不能执行 jps 命令的情况，系统给出的提示是“command not found”。首先，我们要理解，jps 实际上是一个位于 JDK 的 bin 目录下的 Java 命令，其作用是显示当前系统的 Java 进程情况及其 ID 号。jps 相当于 Linux 的进程工具 ps，但不像 pgrep java 或 ps -ef grep java，jps 并不使用应用程序名来查找 JVM 实例。此外，jps 只能查询当前用户的 Java 进程，而不是当前系统中的所有进程。

至于用户为什么不能使用 jps 命令，原因不外两个，一是该命令被意外删除了或者 JDK 没有安装，所以首先检查自己计算机是否已经安装了 JDK；另一个原因是 Java 环境变量没有设置好或者设置不正确，这种情况下，可以先执行带有完整路径的命令 /usr/java/jdk1.7.0_71/bin/jps（这里以本书安装路径为例）；然后检查 .bash_profile 文件中的环境变量设置是否正确；可按照我们前面给出的范例进行设置，即可解决“不能执行 jps 命令”的问题。

在 Hadoop 集群的运维中，系统管理人员还常常使用 Web 界面监视系统状况。例如，在 Master 节点启动 Firefox 浏览器，在浏览器地址栏输入 <http://master:50070/>，可以看到如图 4-34 所示的页面。

同样，在 Firefox 浏览器的地址栏中输入 <http://master:18088/>，可以检查 Yarn 的运行情况，显示的页面如图 4-35 所示。

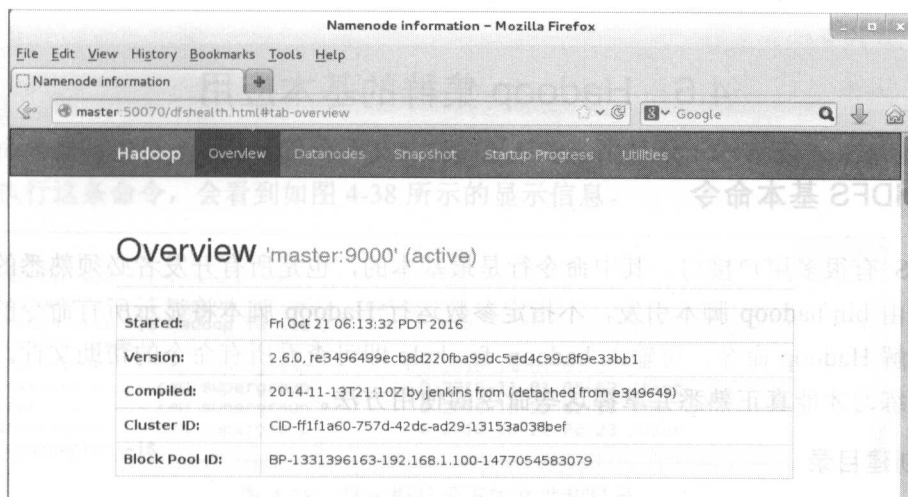


图 4-34 利用 Web 方式检测 Hadoop 运行状态

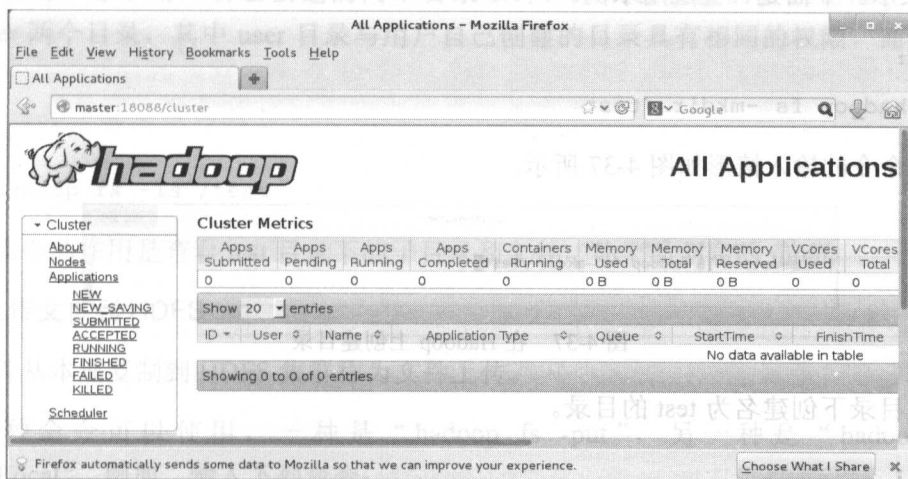


图 4-35 利用 Web 方式检测 Yarn 的运行状态

要关闭 Hadoop 集群，可以使用 `stop-all.sh` 命令，如图 4-36 所示。

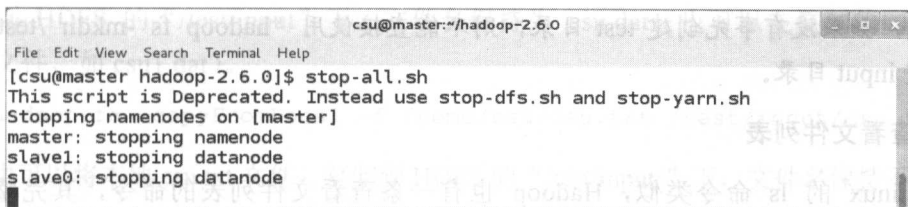


图 4-36 关闭 Hadoop 集群

4.6 Hadoop 集群的基本应用

4.6.1 HDFS 基本命令

HDFS 有很多用户接口，其中命令行是最基本的，也是所有开发者必须熟悉的。所有命令行均由 `bin/hadoop` 脚本引发，不指定参数运行 Hadoop 脚本将显示所有命令的描述。要完整了解 Hadoop 命令，可输入 `hadoop fs -help` 即可查看所有命令的帮助文件。不过，只有通过练习才能真正熟悉并掌握这些命令的使用方法。

1. 创建目录

创建目录用 `-mkdir` 命令。在 Hadoop 上创建目录与在 Linux 上创建目录类似，根目录用 “/” 表示。下面是一些应用示例。

例如：

```
hadoop fs -mkdir /test
```

上述命令的输入情形如图 4-37 所示。

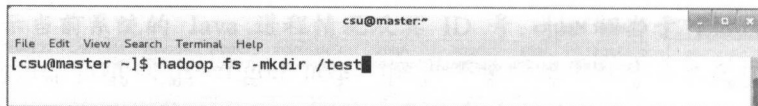


图 4-37 在 Hadoop 上创建目录

在根目录下创建名为 `test` 的目录。

例如：

```
hadoop fs -mkdir /test/input
```

上述命令是在 `test` 目录下创建名为 `input` 的目录。

注意，如果没有事先创建 `test` 目录，则不能直接使用 “`hadoop fs -mkdir /test/input`” 命令创建 `input` 目录。

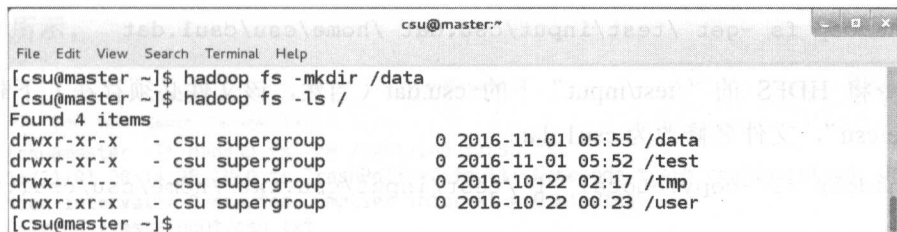
2. 查看文件列表

与 Linux 的 `ls` 命令类似，Hadoop 也有一条查看文件列表的命令，其完整用法是 `hadoop fs -ls <args>`，其中 `<args>` 表示可选参数。

例如：

```
hadoop fs -ls /
```

上述命令显示根目录下的子目录和文件。如果事先在根目录下创建了 `test` 和 `data` 两个目录，执行这条命令，会看到如图 4-38 所示的显示信息。



```

csu@master:~
File Edit View Search Terminal Help
[csu@master ~]$ hadoop fs -mkdir /data
[csu@master ~]$ hadoop fs -ls /
Found 4 items
drwxr-xr-x - csu supergroup 0 2016-11-01 05:55 /data
drwxr-xr-x - csu supergroup 0 2016-11-01 05:52 /test
drwx----- - csu supergroup 0 2016-10-22 02:17 /tmp
drwxr-xr-x - csu supergroup 0 2016-10-22 00:23 /user
[csu@master ~]$

```

图 4-38 显示根目录下的文件和目录

可以看到，除了用户自己创建的两个目录以外，系统在安装的时候也已经创建了 `tmp` 和 `user` 两个目录，其中 `user` 目录与用户自己创建的目录具有相同的权限，而 `tmp` 则不同。

例如：

```
hadoop fs -ls /test
```

这条命令的作用是查看 `test` 目录下的子目录和文件，如上面创建的 `input`。

3. 上传文件到 HDFS

将文件从本地复制到 HDFS 集群称为文件上传。

有两种命令可以使用，一种是 “`hadoop fs -put`”，另一种是 “`hadoop fs -copyFromLocal`”。例如，输入下列命令：

```
hadoop fs -put /home/csu/csu.dat /test/input/csu.dat
```

回车后，将本地 Linux 用户 `csu` 的文件 “`/home/csu/csu.dat`”（当然，该文件必须存在）上传到 HDFS 的 “`/test/input`” 下，文件名保持为 `csu.dat`（也可以在复制的同时重新命名目标文件，如 `csu1.dat`）。

```
hadoop fs -copyFromLocal -f /home/csu/csu.txt /test/input/csu.txt
```

命令也可以将本地 `csu.txt` 文件，复制到 HDFS 的 “`/test/input/`” 下，文件名保持不变。

上述命令中也可以使用相对路径。

4. 下载文件到本地

将文件从 HDFS 集群复制到本地称为文件下载。

有两种命令可以使用，一种是“`hadoop fs -get`”，另一种是“`hadoop fs -copyToLocal`”。举例如下：

```
Hadoop fs -get /test/input/csu.dat /home/csu/csu1.dat
```

此命令将 HDFS 的“`/test/input`”下的 `csu.dat`（当然，该文件必须存在）下载到本地的“`/home/csu`”，文件名修改为 `csu1.dat`。

```
hadoop fs -copyToLocal -f /test/input/csu.txt /home/csu/csu1.txt
```

此命令也可以将 HDFS 的“`/test/input`”下的 `csu.txt` 文件下载到本地的“`/home/csu`”，文件名修改为 `csu1.txt`。

5. 查看 HDFS 文件内容

可以用 `hadoop fs -text`、`hadoop fs -cat`、`hadoop fs -tail` 等不同参数形式查看 HDFS 集群中的文件内容。当然，只有文本文件的内容才可以看清楚，其他文件显示的是乱码。

例如：

```
hadoop fs -cat /test/input/csu.txt
```

此命令将 HDFS 中“`/test/input`”下的 `csu.txt`（当然，该文件必须存在）显示在终端上，如图 4-39 所示。

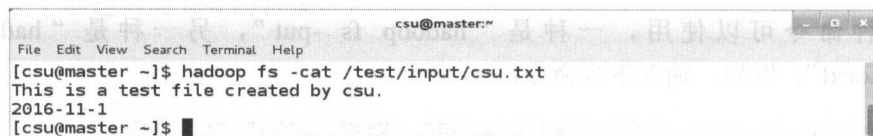


图 4-39 查看 HDFS 中的文件内容

```
hadoop fs -copyToLocal -f /test/input/csu.txt /home/csu/csu1.txt
```

此命令也可以将 HDFS 的“`/test/input`”下的 `csu.txt` 文件下载到本地的“`/home/csu`”，文件名修改为 `csu1.txt`。

将命令行“`hadoop fs -cat /test/input/csu.txt`”中的 `cat` 用 `text` 或 `tail` 代替，看到的结果相同。

6. 删除 HDFS 文件

可以用“`hadoop fs -rm`”删除 HDFS 集群中的文件。例如：

```
hadoop fs -rm /test/input/csu.txt
```

此命令将 HDFS 中“`/test/input`”下的 `csu.txt`（当然，该文件必须存在）删除掉，如图 4-40 所示。

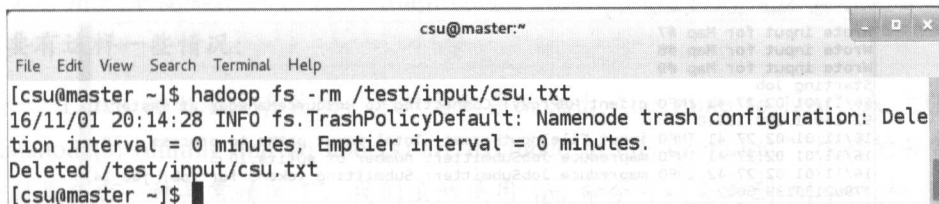


图 4-40 删除 HDFS 中的文件

注意事项

低版本的 HDFS 存在 `hadoop fs` 和 `hadoop dfs` 两种形式的 `hadoop` 命令行，但新版本建议只使用前者。

4.6.2 在 Hadoop 集群中运行程序

下面我们在 Hadoop 集群上运行一个 MapReduce 程序，以帮助读者获得分布式计算的基本印象。

在安装 Hadoop 的时候，系统给用户提供了一些 MapReduce 示例程序，其中有一个典型的用于计算圆周率 π 的 Java 程序包，现在我们将投入运行。

该 jar 包文件的位置和文件名是：

```
~/hadoop-2.6.0/share/hadoop mapreduce/hadoop-mapreduce-examples- 2.6.0.jar
```

我们在终端输入如下的运行命令：

```
hadoop jar ~/hadoop-2.6.0/share/ hadoop/ mapreduce/ hadoop-
mapreduce-examples-2.6.0.jar pi 10 10
```

输入情况如图 4-41 所示。回车后即可开始运行。其中 `pi` 是类名，后面跟了两个 10，它们是运行参数，第一个 10 表示 Map 次数，第二个 10 表示随机生成点的次数（与计算原理有关）。

如

值得指出的是,执行 Hadoop MapReduce 程序,是验证 Hadoop 系统是否正常启动的最后一个环节。实际上,即使通过 jps 和 Web 方式验证了系统已经启动,并且能够查看到状态信息,也不一定意味着系统可以正常工作。例如,防火墙没有关闭的话,MapReduce 程序执行不会成功。

注意事项

经常有用户不能如愿执行上述 MapReduce 程序,所遇到的错误提示也各不相同,主要有这样一些情况:

第一, Linux 防火墙没有关闭,用户看到系统抛出“no route to host”或“`There are 0 datanode(s) running and no node(s) are excluded in this operation.`”这样的异常信息。特别是在后一种异常情况下,我们虽然使用 jps 命令可以查看到 Master 和 Slave 上的 NameNode、DataNode 等进程都存在,但是如果使用 `http://master:50070/` 查看 Hadoop 状态,就会发现没有活动的 DataNode 节点,这就是无法执行 MapReduce 程序的原因。要消除上述异常,就必须关闭 Linux 防火墙,而且所有节点的防火墙都要关闭,然后重新启动 Hadoop 即可。关于关闭防火墙的具体办法,请参见第 3 章中有关内容。

第二,系统不稳定。有时候系统因为某种原因处于一种不稳定状态,也可以导致执行失败,一般可以简单地重新执行一下命令即可。

第三,系统处于安全模式,这时系统会给出提示“`system is in safe mode`”。这时,我们只要执行如下命令:

```
hadoop dfsadmin -safemode leave
```

即可关闭安全模式。离开安全模式后,就可以正常执行 hadoop 命令了。

上述几种异常有时候可以重叠在一起出现,这时候就需要逐个解决问题。

Hadoop 是一种开源系统,且处于不断进化中,出现各类问题是很正常的。从某种意义上讲,开发和应用 Hadoop 大数据应用系统,就是一个不断面对各种问题、需要持续努力和耐心去应对的过程。

4.7 本章小结

本章主要介绍了 Hadoop HDFS 的架构、工作原理,以及 Hadoop 安装、配置、启

动、运行简单程序。读者特别注意，安装 Hadoop 以后，我们采用了三种方式来验证系统是否安装成功：第一种是通过 `jps` 命令，分别在 Master 节点和 Slave 节点上查看 NameNode 和 DataNode 进程是否启动；第二种是通过浏览器访问系统提供的 Web 监控信息；第三种是运行一个 MapReduce 程序，如典型的计算圆周率程序 Pi。

必须指出，如果这三个验证都通过，才能说明系统安装成功，如果运行程序不能成功，说明系统还存在问题，如防火墙没有关闭等。

MapReduce 是 Hadoop 系统中最重要计算引擎，它不仅直接支持交互式应用、基于程序的应用，而且还是 Hive 等组件的基础。MapReduce v2（也就是 Yarn）则进一步提升了该计算引擎的性能和通用性。本章主要介绍基于 MapReduce 的 Java 程序设计方法。

5.1 MapReduce 程序的概念

5.1.1 基本编程模型

MapReduce 采取了分而治之的基本思想，将一个大的作业分解成若干小的任务，提交给集群的多台计算机处理，这样就大大提高了完成作业的效率。在 Hadoop 平台上，MapReduce 框架负责处理并行编程中分布式存储、工作调度、负载均衡、容错及网络通信等复杂工作，把处理过程高度抽象为两个函数：Map 和 Reduce。Map 负责把作业分解成多个任务，Reduce 负责把分解后多任务处理的结果汇总起来。

在 Hadoop 中，用于执行 MapReduce 作业的机器角色有两个：JobTracker 和 TaskTracker。JobTracker 用于调度作业，TaskTracker 用于跟踪任务的执行情况。一个 Hadoop 集群只有一个 JobTracker。

需要注意的是，用 MapReduce 来处理的数据集必须具备这样的特点：数据集可以分解成许多小的数据集，而且每一个小数据集都可以完全独立地并行处理。

最简单的 MapReduce 应用程序至少包含 3 个部分：一个 Map 函数、一个 Reduce 函数和一个 main 函数。在运行一个 MapReduce 程序的时候，整个处理过程被分为两个阶段：Map

阶段和 Reduce 阶段，每个阶段都是用键值对 (key/value) 作为输入 (Input) 和输出 (Output)。main 函数则将作业控制和文件输入/输出结合起来，是 MapReduce 程序的入口。

用户自定义的 Map 函数接收一个输入的 key/value 对值，然后产生一个中间 key/value 对值的集合，接着 MapReduce 把这个中间结果中的所有具有相同 key 的 value 值集合在一起，最后传递给 Reduce 函数。

用户自定义的 Reduce 函数接受一个中间 key 值和相关的 value 值的集合之后，立即合并这些 value 值，从而形成一个较小的 value 值的集合。一般情况下，Reduce 函数每次调用只产生 0 或 1 个输出 value 值。通常经过一个迭代器把中间 value 值提供给 Reduce 函数，这样就可以处理无法全部放入内存中的大量 value 值的集合了。

在 MapReduce 程序中计算的数据可以来自多个数据源，如本地文件、HDFS、数据库等。最常用的是 HDFS，因为可以利用 HDFS 的高吞吐性能读取大规模的数据进行计算；同时，在计算完成后，也可以将数据存储到 HDFS。MapReduce 读取 HDFS 数据或者存储数据到 HDFS 中比较简单。当 MapReduce 运行 Task 时，会基于用户编写的业务逻辑进行读取或存储数据。

5.1.2 计算过程分析

下面我们以 WordCount 程序为例，分析一下 MapReduce 程序的执行过程。在开始描述之前，我们来看看作业是什么。设输入若干文本文件，我们的目标是统计所有文件中每个单词出现的次数 (频次)。以如图 5-1 所示的两个输入文件为例，直观地看，Bye 出现 2 次，Hello 出现 2 次，World 出现 4 次。WordCount 程序就是要实现这样的统计。

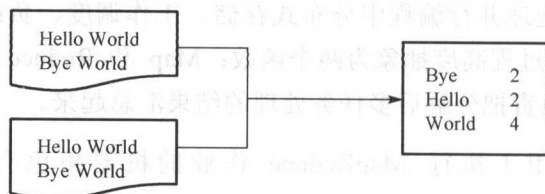


图 5-1 WordCount 作业的目标

结合上面的示例，我们来分析 MapReduce 程序的执行过程，可分为下面几个步骤。

1. 拆分输入数据

拆分数据 (Split) 属于 Map 的输入阶段，系统会逐行读取文件的数据，得到一系列的 (key/value)，如图 5-2 所示。

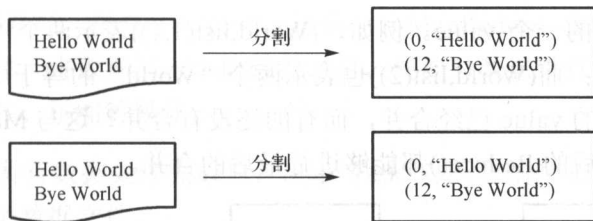


图 5-2 拆分输入数据

注意，如果只有一个文件，且很小，这时系统只分配一个 Split；如果有多个文件，或者文件很大，系统会根据需要分配多个 Split。这一步由 MapReduce 框架自动完成。图 5-2 中的数值是一个偏移量（即 key 值），它是包括回车符在内的字符数。

2. 执行 Map 方法

分割完成后，系统会将分割好的（key/value）对交给用户定义的 Map 方法进行处理，生成新的（key/value）对，如图 5-3 所示。

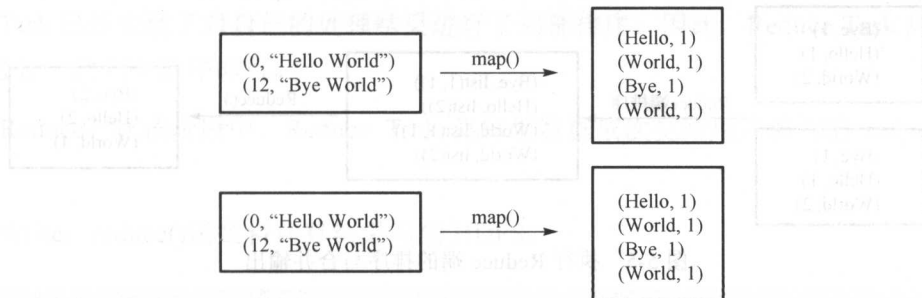


图 5-3 执行用户定义的 Map 方法

3. 排序与合并处理

系统在得到 Map 方法输出的（key/value）对后，Mapper 会将它们按照 key 值进行排序，并执行 Combine 过程，将 key 值相同的 value 值累加，得到 Mapper 的最终输出结果。如图 5-4 所示。

4. Reduce 阶段的排序与合并

Reducer 先对从 Mapper 接收的数据进行排序，再交由用户自定义的 Reduce 方法进行处理，得到新的（key/value）对，并作为 WordCount 的结果输出，如图 5-5 所示。

图 5-5 的中间部分出现的 list（数据列表）是与 MapReduce 内部合并机制（Combine）有关的一个函数，其中 list(1,1)表示尚未合并的两个等于 1 的 value，而 list(2)

则表示已经合并为 2 的一个 value。例如, (World,list(1,1))表示两个“World”的等于 1 的 value, 但还没有合并; 而(World,list(2))也表示两个“World”的等于 1 的 value, 但已经合并。为什么会出现有的 value 已经合并, 而有的还没有合并? 这与 MapReduce 的内部资源管理机制有关, 但最后的 Reduce()都能够进行最后的合并。



图 5-4 执行 map 阶段的排序与合并

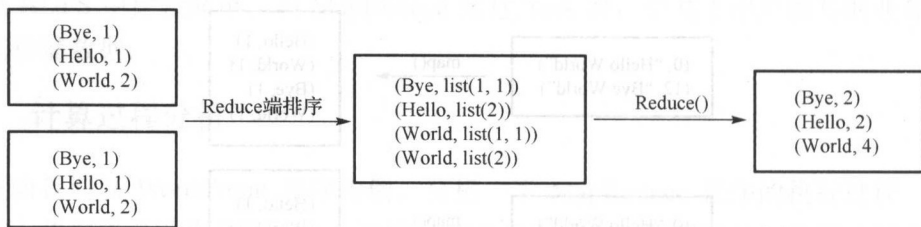


图 5-5 执行 Reduce 端的排序与合并输出

实际上, 上面的分析是一种直观的理解。如果结合源代码分析, 就可以看到处理流程的更多细节。

(1) Map 阶段可以概括为 5 个步骤。

① Read: Map Task 通过用户编写的 RecordReader, 从输入 InputSplit 中解析出一个一个的 (key/value)。

② Map: 该步骤主要将解析出的 (key/value) 交给用户编写的 Map 函数处理, 并产生一系列新的 (key/value)。

③ Collect: 在用户编写的 Map 函数中, 数据处理完成后, 一般会调用 OutputCollector.collect()收集结果。在该函数内部, 它会将生成 (key/value) 分片 (通过 Partitioner), 并写入一个环形内存缓冲区中。

④ **Spill**: 即所谓溢写, 指当环形缓冲区填满后, MapReduce 会将数据写到本地磁盘上, 生成一个临时文件。将数据写入本地磁盘之前, 先要对数据进行一次本地排序, 并在必要时对数据进行合并、压缩等操作。

⑤ **Combine**: 当所有数据处理完成后, Map Task 对所有临时文件进行一次合并, 以确保最终只会生成一个数据文件。

(2) Reduce 阶段可分解为如下 5 个步骤。

① **Shuffle**: 也称为 Copy 阶段。Reduce Task 从各个 Map Task 上远程复制一片数据, 并针对某一片数据进行判断, 如果其大小超过一定阈值, 则写到磁盘上, 否则直接放到内存中。

② **Merge**: 在远程复制的同时, Reduce Task 启动了两个后台线程对内存和磁盘上的文件进行合并, 以防止内存使用过多或者磁盘上文件过多。

③ **Sort**: 按照 MapReduce 语义, 用户编写的 Reduce 函数输入数据是按 key 进行聚集的一组数据。为了将 key 相同的数据聚在一起, Hadoop 采用了基于排序的策略。由于各个 Map Task 已经实现了对自己的处理结果进行了局部排序, 因此, Reduce Task 只需对所有数据进行一次归并排序即可。

④ **Reduce**: 在该阶段中, Reduce Task 将每组数据依次交给用户编写的 `reduce()` 函数处理。

⑤ **Write**: `reduce()` 函数将计算结果写到 HDFS。

5.2 深入理解 Yarn

5.2.1 Yarn 的基本架构

Yarn 的基本思想是将 JobTracker 的资源管理和作业的调度/监控两大主要职能拆分为两个独立的进程: 一个全局的 Resource Manager 和与每个应用对应的 Application Master (AM)。Resource Manager 和每个节点上的 Node Manager (NM) 组成了全新的通用操作系统, 以分布式的方式管理应用程序。图 5-6 给出了 Yarn 的系统架构示意。

Resource Manager 拥有为系统中所有应用分配资源的决定权。与之相关的是应用程序的 Application Master, 负责与 Resource Manager 协商资源, 并与 Node Manager 协同工作来执行和监控任务。

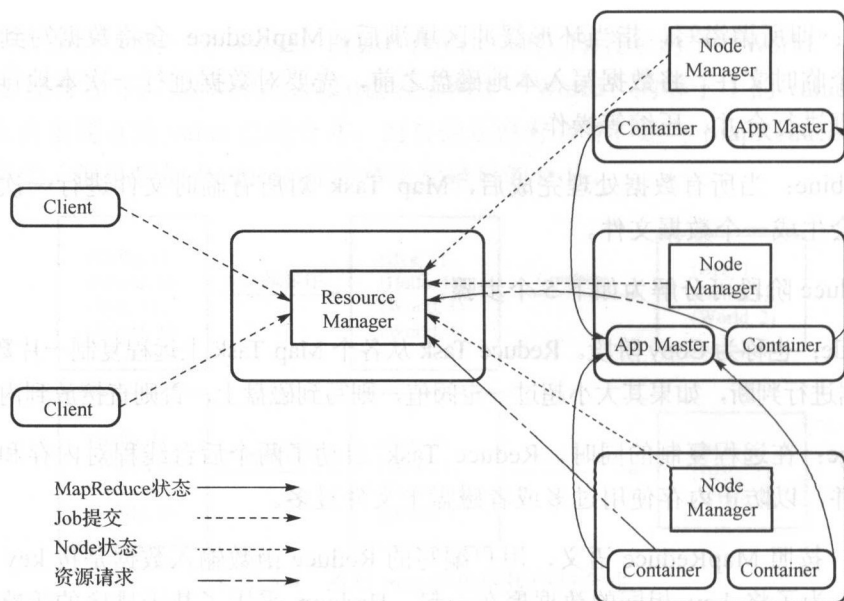


图 5-6 Yarn 系统架构

Resource Manager 有一个可插拔的调度器组件 **Scheduler**，它负责为运行中的各种应用分配资源，分配时会受到容量、队列及其他因素的制约。**Scheduler** 是一个纯粹的调度器，不负责应用程序的监控和状态跟踪，也不保证在应用程序失败或者硬件失败的情况下对 **Task** 的重启。**Scheduler** 基于应用程序的资源的需求来执行其调度功能，使用了称为资源 **Container** 的抽象概念，其中包括了多种资源维度，如内存、CPU、磁盘及网络。

Node Manager 是与每台机器对应的从属进程 (Slave)，负责启动应用程序的 **Container**，监控它们的资源使用情况 (CPU、内存、磁盘和网络)，并且报告给 **Resource Manager**。

每个应用程序的 **Application Master** 负责与 **Scheduler** 协商合适的 **Container**，跟踪应用程序的状态，以及监控它们的进度。从系统的角度讲，**Application Master** 也是以一个普通 **Container** 的身份运行的。

在新的 Yarn 系统下，**MapReduce** 的一个关键思想是，确保与现有 **MapReduce** 应用和用户兼容，也就是重用现有的 **MapReduce** 框架。

下面我们分别介绍 Yarn 的各个组件。

1. Resource Manager

Yarn **Resource Manager** 是一个纯粹的调度器，它负责整个系统的资源管理和分配。它

本身主要由两个组件构成：调度器（Scheduler）和应用程序管理器（Applications Manager, AM）。

调度器根据容量、队列等限制条件（如每个队列分配一定的资源，最多执行一定数量的作业等），将系统中的资源分配给各个正在运行的应用程序。需要注意的是，该调度器是一个“纯调度器”，它不再从事任何与具体应用程序相关的工作，例如不负责监控或者跟踪应用的执行状态等，也不负责重新启动因应用执行失败或者硬件故障而产生的失败任务，这些均交由应用程序相关的 Application Master 完成。调度器仅根据各个应用程序的资源需求进行资源分配，而资源分配单位用一个抽象概念“资源容器”（Resource Container, 简称 Container）表示，Container 是一个动态资源分配单位，它将内存、CPU、磁盘、网络等资源封装在一起，从而限定每个任务使用的资源量。此外，该调度器是一个可插拔的组件，用户可根据自己的需要设计新的调度器，Yarn 提供了多种直接可用的调度器，如公平调度器（Fair Scheduler）和容量调度器（Capacity Scheduler）。

应用程序管理器负责管理整个系统中所有应用程序，包括应用程序提交、与调度器协商资源以启动 Application Master、监控 Application Master 运行状态并在失败时重新启动它等。

2. Application Master

Yarn 的另一个重要新概念就是 Application Master。Application Master 实际上是特定框架库的一个实例，负责与 Resource Manager 协商资源，并和 Resource Manager 协同工作来执行和监控 Container，以及它们的资源消耗。它有责任与 Resource Manager 协商并获取合适的资源 Container，跟踪它们的状态，以及监控其进展。

Application Master 和应用是相互对应的，它主要有以下职责：

- 与调度器协商资源；
- 与 Node Manager 合作，在合适的 Container 中运行对应的组件 Task，并监控这些 task 执行；
- 如果 Container 出现故障，Application Master 会重新向调度器申请其他资源；
- 计算应用程序所需的资源量，并转化成调度器可识别的协议信息包；
- 在 Application Master 出现故障后，应用管理器会负责重启它，但由 Application Master 自己从之前保存的应用程序执行状态中恢复应用程序。

在真实环境下，每一个应用都有自己的 Application Master 实例。然而，为一组应用提供一个 Application Master 是完全可行的，如 Pig 或者 Hive 的 Application Master。另

外，这个概念已经延伸到了管理长时间运行的服务，它们可以管理自己的应用。例如，通过一个特殊的 HBaseApp Master 在 Yarn 中启动 HBase。

3. Node Manager

Node Manager 是每个节点的框架代理。它负责启动应用的 Container，监控 Container 的资源使用（包括 CPU、内存、硬盘和网络带宽等），并把这些用信息汇报给调度器。应用对应的 Application Master 负责通过协商从调度器处获取 Container，并跟踪这些 Container 的资源状态和应用执行的情况。集群每个节点上都有一个 Node Manager，它主要负责：

- 为应用启用调度器，以分配给应用的 Container；
- 已启用的 Container 不会使用超过分配的资源量；
- 为 Task 构建 Container 环境，包括二进制可执行文件.jar 等；
- 为所在的节点提供一个管理本地存储资源的简单服务。

应用程序可以继续使用本地存储资源，即使它没有从 Resource Manager 处申请。例如，MapReduce 可以利用这个服务存储 Map Task 的中间输出结果，并将其 Shuffle 给 Reduce Task。

Yarn 的应用资源模型是一个通用的设计。一个应用（通过 Application Master）可以请求非常具体的资源，包括：

- 资源名称（包括主机名称、机架名称，以及可能的复杂的网络拓扑）；
- 内存量；
- CPU（核数/类型）；
- 其他资源，如 Disk、Network、I/O、GPU 等。

4. Resource Request 和 Container

Yarn 被设计成可以允许应用程序（通过 Application Master）以共享的、安全的，以及多用租户的方式使用集群的资源。它也会感知集群的网络拓扑，以便可以有效地调度，以及优化数据访问（即尽可能地为应用减少数据移动）。

为了实现这些目标，位于 Resource Manager 内的中心调度器保存了应用程序的资源请求的信息，以帮助它为集群中的所有应用做出更优的调度决策。由此引出了 Resource Request，以及由此产生的 Container 概念。

本质上，一个应用程序可以通过 Application Master 请求特定的资源需求来满足它的

资源需要。调度器会分配一个 Container 来响应资源需求，用于满足由 Application Master 在 Resource Request 中提出的需求。

Resource Request 具有以下形式：

<资源名称，优先级，资源需求，Container 数>

其中：

- 资源名称是资源期望所在的主机名、机架名，用*表示没有特殊要求。未来可能支持更加复杂的拓扑，例如一个主机上的多个虚拟机，更复杂的网络拓扑等。
- 优先级是应用程序内部请求的优先级（而不是多个应用程序之间），优先级会调整应用程序内部各个 Resource Request 的次序。
- 资源需求是需要的资源量，如内存大小和 CPU 时间（目前 Yarn 仅支持内存和 CPU 两种资源维度）。
- Container 数表示需要这样的 Container 的数量，它限制了使用该 Resource Request 指定的 Container 的总数。

本质上，Container 是一种资源分配形式，是 Resource Manager 为 Resource Request 成功分配资源的结果。Container 为应用程序授予在特定主机上使用资源（如内存、CPU 等）的权利。

Application Master 必须取走 Container，并交给 Node Manager，Node Manager 会利用相应的资源来启动 Container 的任务进程。出于安全考虑，Container 的分配要以一种安全的方式进行验证，来保证 Application Master 不能伪造集群中的应用。

关于 Container，存在一个规范。如前所述，Container 只有使用服务器（Node Manager）上指定资源的权利，Application Master 必须向 Node Manager 提供更多信息来启动 Container。与现有的 MapReduce 不同，Yarn 允许应用程序启动任何程序，而不仅限于 Java 应用程序。

5.2.2 Yarn 的工作流程

经典的 MapReduce（MapReduce v1）的最顶层包含 4 个独立的实体，分别是客户端、JobTracker、TaskTracker，以及分布式文件系统。Yarn 将 JobTracker 的职能划分为多个独立的实体，从而改善经典 MapReduce 面临的扩展瓶颈问题。JobTracker 负责作业调度和任务进度监视、追踪任务、重启失败或过慢的任务及进行任务登记，例如维护计数器总数。

Yarn 将这两种角色划分为两个独立的守护进程：管理集群上资源使用的资源管理器

和管理集群上运行任务生命周期的应用管理器。基本思路是：应用服务器与资源管理器协商集群的计算资源——容器（每个容器都有特定的内存上限），在这些容器上运行特定应用程序的进程。容器由集群节点上运行的节点管理器监视，以确保应用程序使用的资源不会超过分配给它的资源。

与 JobTracker 不同，应用的每个实例（这里指一个 MapReduce 作业）都有一个特定的应用 Master，它运行在应用的存续期间。这种方式实际上和最初 Google 的 MapReduce 论文里介绍的方法很相似，该论文描述了 Master 进程如何协调在一组 Worker 上运行的 Map 任务和 Reduce 任务。

因此，Yarn 比 MapReduce v1 更具一般性，实际上 MapReduce v1 只是 Yarn 应用的一种形式。有很多其他的 Yarn 应用（例如能够在集群中的一组节点上运行脚本的分布式 Shell）及其他正在开发的程序。Yarn 设计的精妙之处在于不同的 Yarn 应用可以在同一个集群上共存，这大大提高了可管理性和集群的利用率。

此外，用户甚至有可能在同一个 Yarn 集群上运行多个不同版本的 MapReduce，这使得 MapReduce 升级过程更容易管理。注意，MapReduce 的某些部分（如作业历史服务器和 Shuffle 处理器），以及 Yarn 本身仍然需要在整个集群上升级。

Yarn 上的 MapReduce 比经典的 MapReduce 包括更多的实体，总体上，Yarn 的工作流程包括 5 个步骤：

第一，客户端提交 MapReduce 作业。

第二，Yarn 资源管理器负责协调集群上计算资源的分配。

第三，Yarn 节点管理器（Node Manager）负责启动和监视集群中机器上的计算容器（Container）。

第四，应用程序的 Master 负责协调运行 MapReduce 作业的任务，它和 MapReduce 任务在容器中运行，这些容器由资源管理器分配对节点管理器进行管理。

第五，分布式文件系统（HDFS）用来与其他实体间共享作业文件。

5.3 在 Linux 平台安装 Eclipse

为了在 Hadoop 平台上开发应用程序，我们需要安装有效的集成开发环境（IDE）。本节介绍 Eclipse 的安装与应用，第 8 章还将介绍 IDEA。

5.3.1 Eclipse 简介

Eclipse 是著名的跨平台的自由集成开发环境 (IDE)，最初主要用来 Java 语言开发。通过安装不同的插件 Eclipse 可以支持不同的计算机语言，如 C++ 和 Python 等开发工具。Eclipse 本身只是一个框架平台，但是众多插件的支持使得 Eclipse 拥有其他功能相对固定的 IDE 软件很难具有的灵活性。许多软件开发商以 Eclipse 为框架开发自己的 IDE。

Eclipse 最初由 OTI 和 IBM 两家公司的 IDE 产品开发组创建，起始于 1999 年 4 月。IBM 提供了最初的 Eclipse 代码基础，包括 Platform、JDT 和 PDE。Eclipse 项目由 IBM 发起，围绕着 Eclipse 项目已经发展成为了一个庞大的 Eclipse 联盟，有 150 多家软件公司参与到 Eclipse 项目中，其中包括 Borland、Rational Software、Red Hat 及 Sybase 等。Eclipse 是一个开放源码项目，它其实是 Visual Age for Java 的替代品，其界面跟先前的 Visual Age for Java 差不多，但由于其开放源码，任何人都可以免费得到，并可以在此基础上开发各自的插件，因此越来越受人们的关注。随后还有包括 Oracle 在内的许多大公司也纷纷加入了该项目，Eclipse 的目标是成为可进行任何语言开发的 IDE 集成者，使用者只需下载各种语言的插件即可。

Eclipse 最初是由 IBM 公司开发的、替代商业软件 Visual Age for Java 的下一代 IDE 开发环境，2001 年 11 月贡献给开源社区，现在它由非营利软件供应商联盟 Eclipse 基金会 (Eclipse Foundation) 管理。2003 年，Eclipse 3.0 选择 OSGi 服务平台规范为运行时架构；2007 年 6 月，稳定版 3.3 发布；2008 年 6 月发布代号为 Ganymede 的 3.4 版；2009 年 6 月发布代号为 Galileo 的 3.5 版；2010 年 6 月发布代号为 Helios 的 3.6 版；2011 年 6 月发布代号为 Indigo 的 3.7 版；2012 年 6 月发布代号为 Juno 的 4.2 版；2013 年 6 月发布代号为 Kepler 的 4.3 版；2014 年 6 月发布代号为 Luna 的 4.4 版；2015 年 6 月项目发布代号为 Mars 的 4.5 版；2016 年 9 月发布 4.6 版。

Eclipse 是一个开放源代码的、基于 Java 的可扩展开发平台。就其本身而言，它只是一个框架和一组服务，用于通过插件构建开发环境。幸运的是，Eclipse 附带了一个标准的插件集，包括 Java 开发工具 (Java Development Kit, JDK)。虽然大多数用户很乐于将 Eclipse 当成 Java 集成开发环境 (IDE) 来使用，但 Eclipse 的目标却不仅限于此。Eclipse 还包括插件开发环境 (Plug-in Development Environment, PDE)，这个组件主要针对希望扩展 Eclipse 的软件开发人员，因为它允许他们构建与 Eclipse 环境无缝集成的工具。由于 Eclipse 中的每样东西都是插件，对于给 Eclipse 提供插件，以及给用户提供一致和统一的集成开发环境而言，所有工具开发人员都具有同等的发挥场所。

这种平等和一致性并不仅限于 Java 开发工具。尽管 Eclipse 是使用 Java 语言开发的，

但它的用途并不限于 Java 语言，例如支持诸如 C/C++、COBOL、PHP、Android 等编程语言的插件已经可用，或预计将会推出。Eclipse 框架还可作为与软件开发无关的其他应用程序类型的基础，如内容管理系统。

5.3.2 安装并启动 Eclipse

读者可以从 <https://www.eclipse.org/downloads/> 下载最新版本的 Eclipse，也可以在本章软件资源文件夹中找到 eclipse-jee-indigo-SR2-linux-gtk-x86_64.tar.gz 文件。请将该文件复制到 Master 节点的 “/home/csu/” 目录下。

进入 “/home/csu/” 目录，执行如下命令，即可解压缩并安装 Eclipse。

```
tar -zxvf eclipse-jee-indigo-SR2-linux-gtk-x86_64.tar.gz
```

执行完毕，系统即创建 “/home/csu/eclipse”，这就是 Eclipse 的主安装目录。

完成上述操作即可启动 Eclipse 进行开发了。

要启动 Eclipse，首先进入 Eclipse 安装目录，然后在 Linux 命令行中输入如下命令：

```
./eclipse
```

回车后系统弹出图 5-7 所示的工作目录配置页面，可以直接选择缺省的 workspace 目录；也可以单击 “Browse...” 按钮选择不同的工作目录。

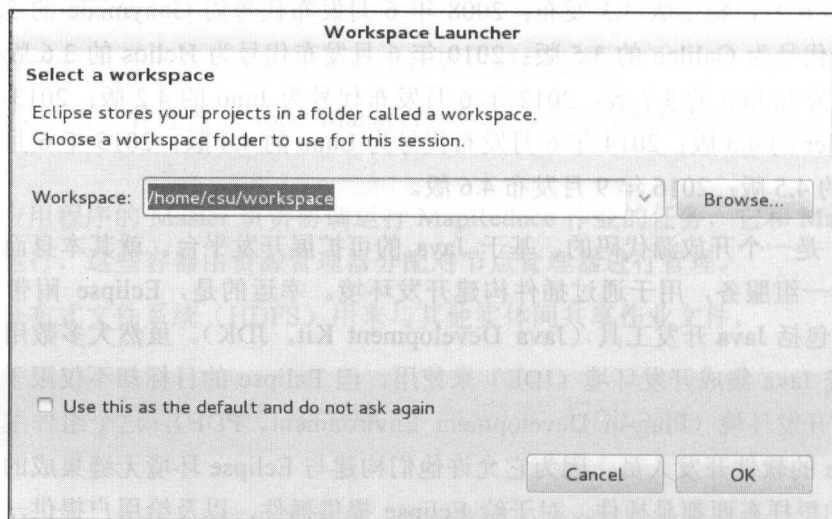


图 5-7 启动 Eclipse 后选择 workspace

然后单击图 5-7 中的 “OK” 按钮，即可进入 Eclipse 的欢迎页面，如图 5-8 所示。

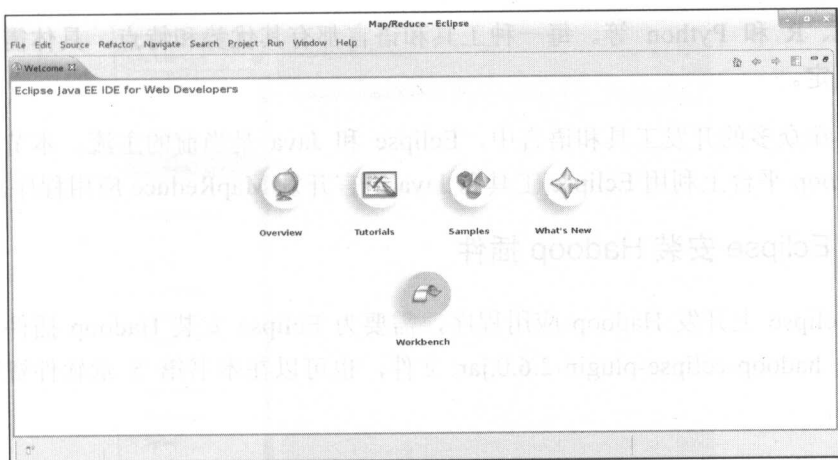


图 5-8 首次启动 Eclipse 的欢迎页面

在图 5-8 中，读者可以单击“Overview”图标了解 Eclipse 的总体介绍，单击“Tutorials”图标了解基本应用方法，等等。要进入程序设计界面，可单击“workbench”图标，如图 5-9 所示。

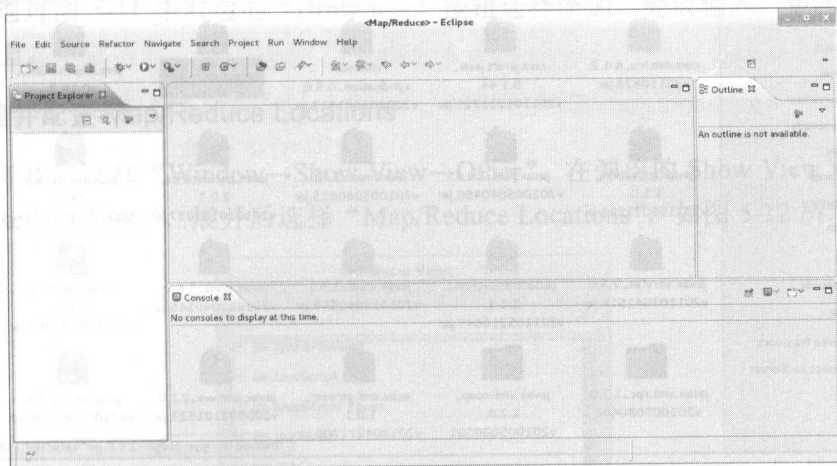


图 5-9 Eclipse 开发主界面

5.4 开发 MapReduce 程序的基本方法

开发 Hadoop 平台上的 MapReduce 应用程序有多种方法。从开发工具上看，我们可以选择 Eclipse、IntelliJ IDEAD（本书后面也有介绍）等。从程序设计语言来看，可以选择

Java、Scala、R 和 Python 等。每一种工具和语言都有其优势和特点，具体需要根据项目的特性来决定。

但是，在众多的开发工具和语言中，Eclipse 和 Java 是当前的主流。本节我们来介绍如何在 Hadoop 平台上利用 Eclipse 工具和 Java 语言开发 MapReduce 应用程序。

5.4.1 为 Eclipse 安装 Hadoop 插件

要在 Eclipse 上开发 Hadoop 应用程序，需要为 Eclipse 安装 Hadoop 插件。读者可以从网络下载 `hadoop-eclipse-plugin-2.6.0.jar` 文件，也可以在本书第 5 章软件资源文件夹中找到该文件。

1. 将 `hadoop-eclipse-plugin-2.6.0.jar` 文件复制到 “`eclipse/plugins/`” 下

请将 `hadoop-eclipse-plugin-2.6.0.jar` 复制到 Master 节点的 “`/home/csu/eclipse/plugins/`” 目录下，如图 5-10 所示。注意，一定要放在 Eclipse 安装目录下的插件子目录中。

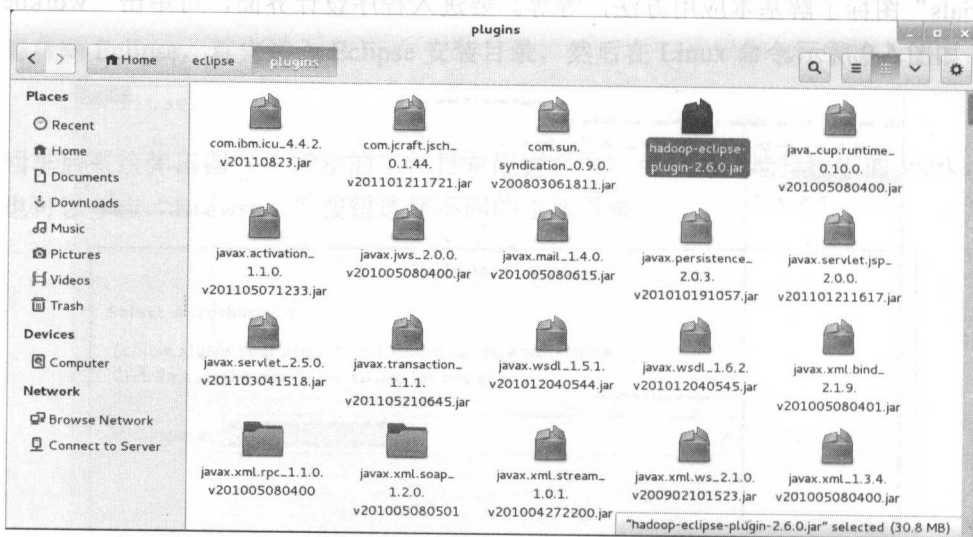


图 5-10 将 `hadoop-eclipse-plugin-2.6.0.jar` 放在 Eclipse 安装目录下的 `plugins` 子目录下

2. 在 Eclipse 中设置 Hadoop 的安装目录

需要重新启动 Eclipse（这样就能自动感知新增的 Hadoop 插件），进入主界面。在主菜单中，选择 “Window→Preferences”，在弹出的 Preferences 对话框中，请点选左边的 Hadoop Map/Reduce，然后在右边的 “Hadoop installation directory” 编辑框中，输入 Hadoop 的安装目录，如 “`/home/csu/hadoop-2.6.0`”，如图 5-11 所示。

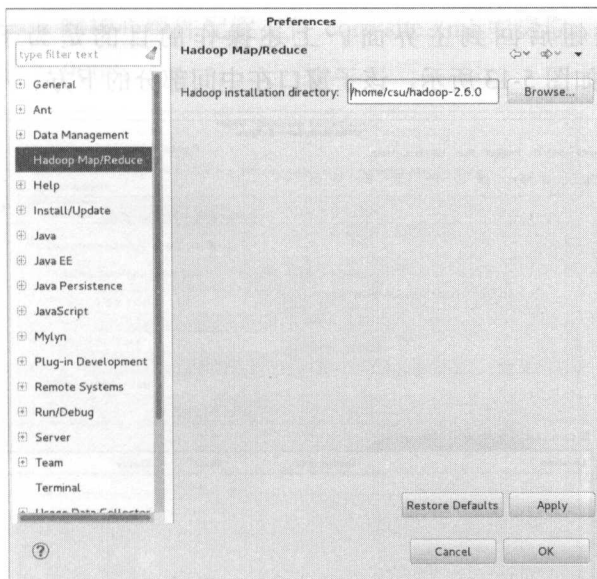


图 5-11 在 Eclipse 的 Preferences 属性中设置 Hadoop 的安装目录

也可以通过图 5-11 中右边的“Browse...”按钮选择添加。完成输入，单击“OK”按钮回到主界面。

3. 创建并配置 Map/Reduce Locations

在主菜单中，选择“Window→Show View→Other”，在弹出的 Show View 对话框中，找到“MapReduce Tools”，展开后选择“Map/Reduce Locations”，如图 5-12 所示。

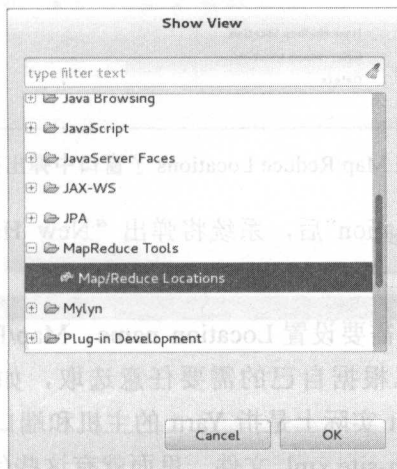


图 5-12 在 Eclipse 的 Show View 对话框中选择 Map/Reduce Locations

单击“OK”按钮后回到主界面。上述操作的目的是为了打开“Map/Reduce Locations”子窗口，如图 5-13 所示，该子窗口在中间部分的下方。

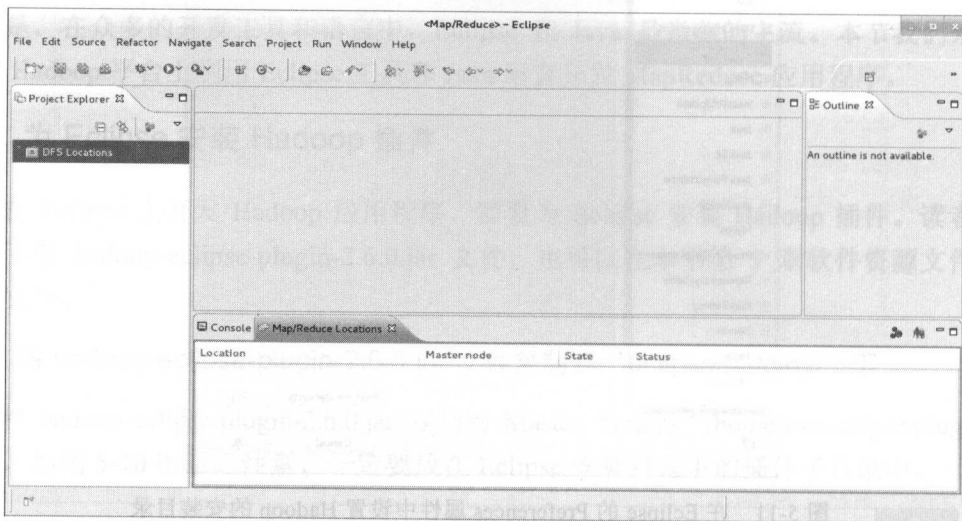


图 5-13 在 Eclipse 打开的 Map/Reduce Locations 子窗口

接着，将鼠标移动到“Map/Reduce Locations”子窗口内，单击鼠标右键，系统弹出菜单列表，请选择“New Hadoop Location”，如图 5-14 所示，我们准备创建并配置一个新的 Hadoop Location。

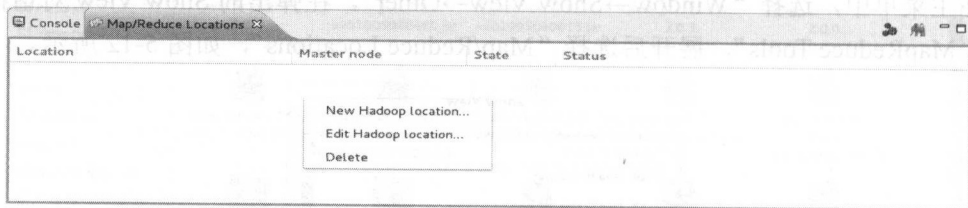


图 5-14 在 Map/Reduce Locations 子窗口中弹出列表式菜单

选择“New Hadoop Location”后，系统将弹出“New Hadoop Location...”设置对话框，如图 5-15 所示。

在 General 中页面，我们需要设置 Location name、Map/Reduce (V2) Master 和 DFS Master。Location name 可以根据自己的需要任意选取，如 CSU_Hadoop。Map/Reduce (V2) Master 的 Host 和 Port 实际上是指 Yarn 的主机和端口，我们需要回顾第 4 章安装 Hadoop 的时候所设置的 yarn-site.xml 文件，里面就有这些信息。为稳妥起见，我们采用 IP 地址代替主机名，如 192.168.1.100，端口号可以取默认的 50020。DFS Master 的 Host

和 Port 则需要从 Hadoop 的核心组件配置文件 `core-site.xml` 中提取。其中, 主机名会被自动设置, 而端口号则设为 9000。

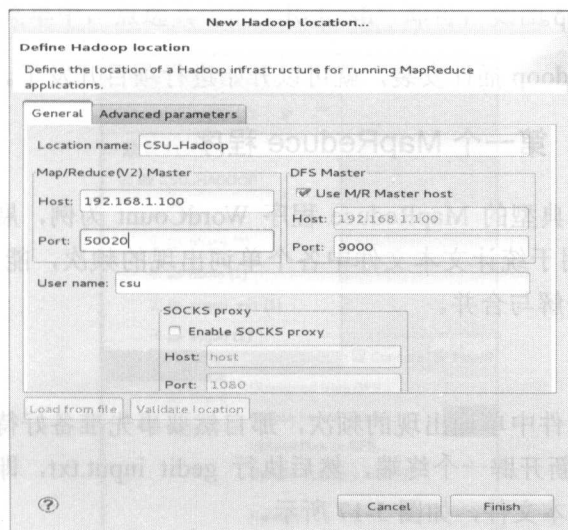


图 5-15 设置 Hadoop Location 属性

单击图 5-15 的“Finish”按钮后, 返回主界面。如果上述配置正确, 将会在左边的“Project Explorer”窗口中的 DFS Locations 下看到新增的 Hadoop Location, 并可以展开看到 Hadoop 中的文件目录, 如图 5-16 所示。

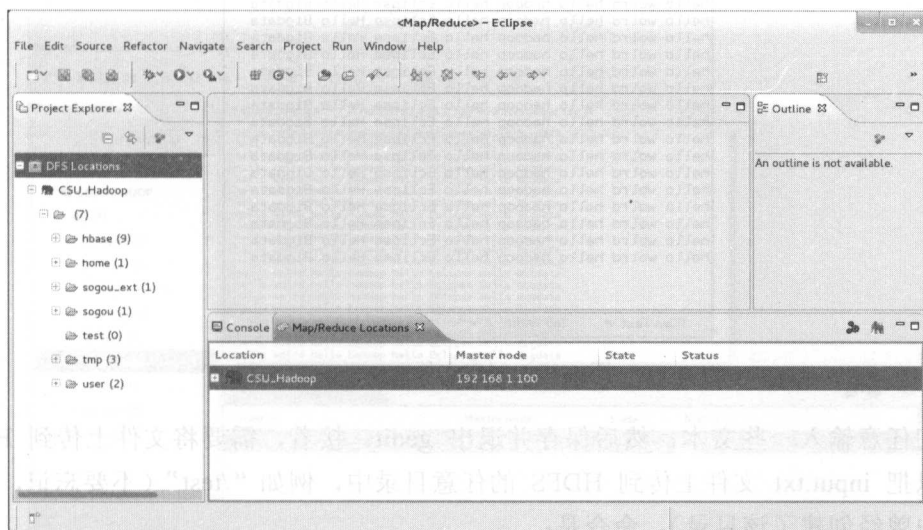


图 5-16 成功创建新的 Hadoop Location

有些用户在进行上述配置后, Eclipse 不能连接 Hadoop, 并给出错误提示, 这时候需要仔细检查配置是否正确。此外, 还要特别注意, 自己计算机的 Hadoop 是否启动了, 必须确保已经启动 Hadoop。

完成 Eclipse 的 Hadoop 插件安装, 就可以开始进行项目开发了。

5.4.2 WordCount: 第一个 MapReduce 程序

本节以 Hadoop 中典型的 MapReduce 程序 WordCount 为例, 展示完整的程序设计过程。WordCount 程序用于统计文本文件中各个单词出现的频次, 能充分体现 MapReduce 计算模型的特点, 即分解与合并。

1. 准备数据文件

既然要统计文本文件中单词出现的频次, 那自然要事先准备好待处理的数据文件。为此, 我们在 Master 上新开辟一个终端, 然后执行 `gedit input.txt`, 即可在当前目录下创建一个名为 `input.txt` 的文本文件, 如图 5-17 所示。

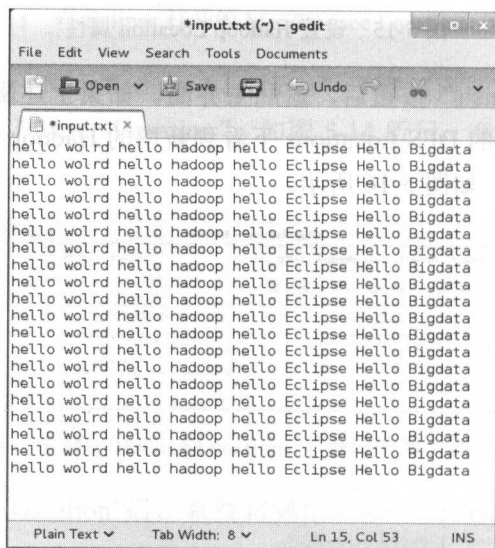


图 5-17 创建文件 input.txt

我们任意输入一些文本, 然后保存并退出 `gedit`。接着, 需要将文件上传到 HDFS。我们可以把 `input.txt` 文件上传到 HDFS 的任意目录中, 例如 `/test` (不要忘记, 我们在第 4 章中曾经创建了该目录), 命令是:

```
hadoop fs -put /home/csu/input.txt /test/
```

上传成功后,可以在 Eclipse 的 DFS Locations 下的 CSU_HADOOP 中看到上传的文件,但必须先手动刷新一下。方法是,选择 CSU_HADOOP 下的 test 文件夹,单击鼠标右键,在弹出的列表式菜单中,点选其中的“Refresh”即可,如图 5-18 所示。

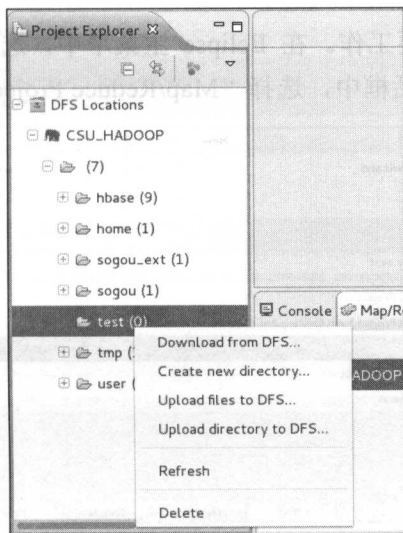


图 5-18 刷新 Hadoop 文件内容

这时系统将显示新上传的文件 input.txt,于是我们双击该文件,其内容就会显示在一个页面内,如图 5-19 所示。

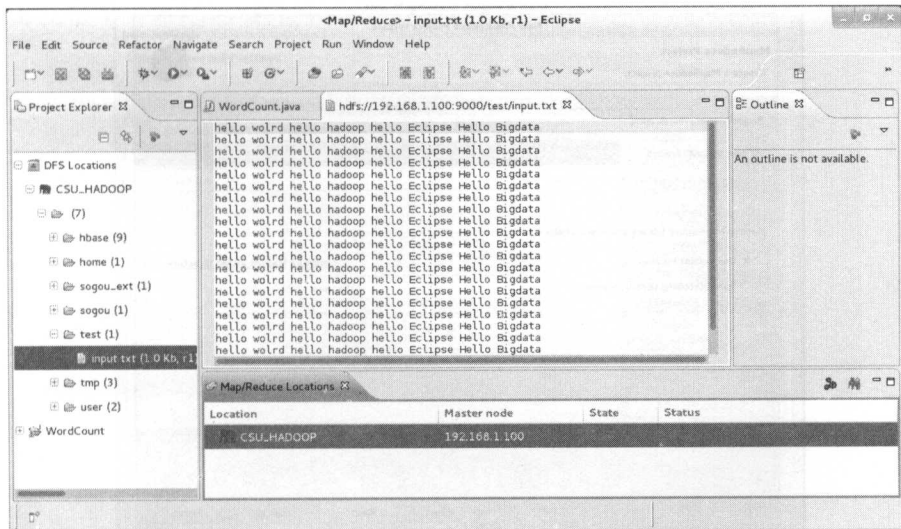


图 5-19 在 Eclipse 中打开 input.txt 文件

实际上, 我们可以在 Eclipse 中进行文件目录创建、文件上传、文件下传、文件或文件夹删除等操作, 但是不能编辑文件内容。

2. 新建项目

显然, 创建新项目是主要工作。在 Eclipse 主菜单中, 选择“File→New→Other”, 在弹出的“Select a wizard”对话框中, 选择“Map/Reduce Project”, 如图 5-20 所示。

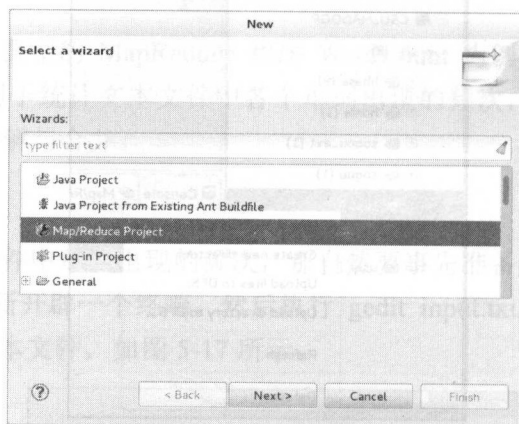


图 5-20 创建一个新的 Map/Reduce Project

选择后, 单击图 5-20 中的“Next”按钮, 在随后出现的对话框中, 我们需要给项目命名, 例如这里的 WordCount, 如图 5-21 所示。

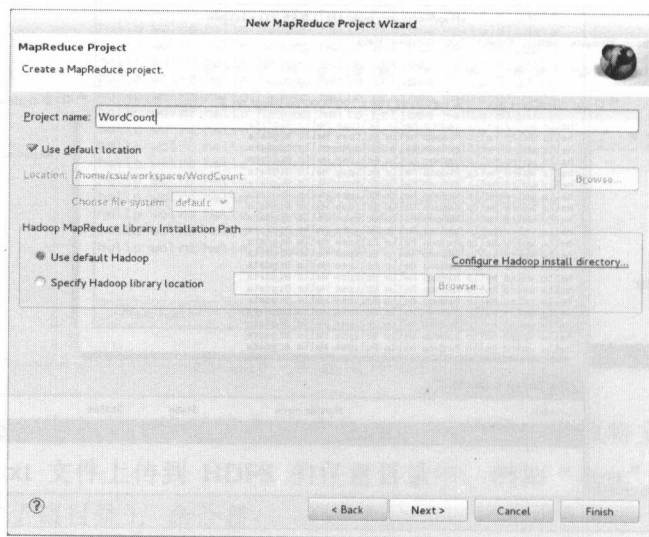


图 5-21 给新的 Map/Reduce Project 取名

接着单击图 5-21 中的“Next”按钮，系统会弹出 Java 运行环境设置对话框，如图 5-22 所示。

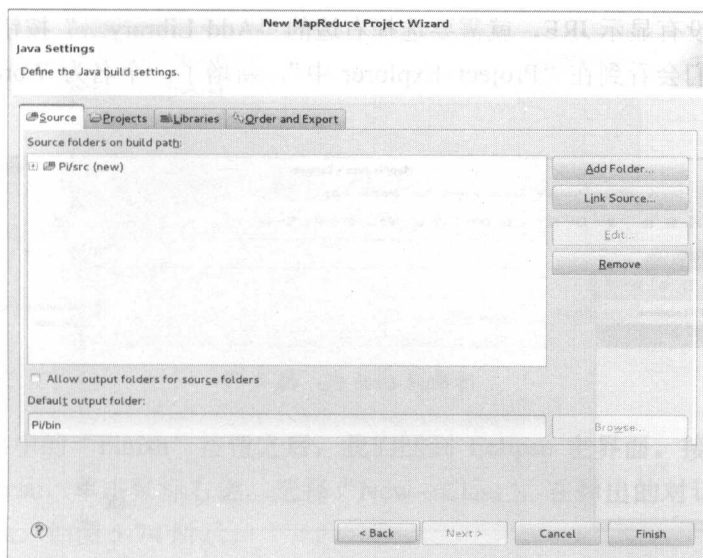


图 5-22 Java 设置对话框

在图 5-22 中，请选择“Libraries”页面，我们看到，系统给出了已经安装的 JRE (Java 运行时环境)，如图 5-23 所示。

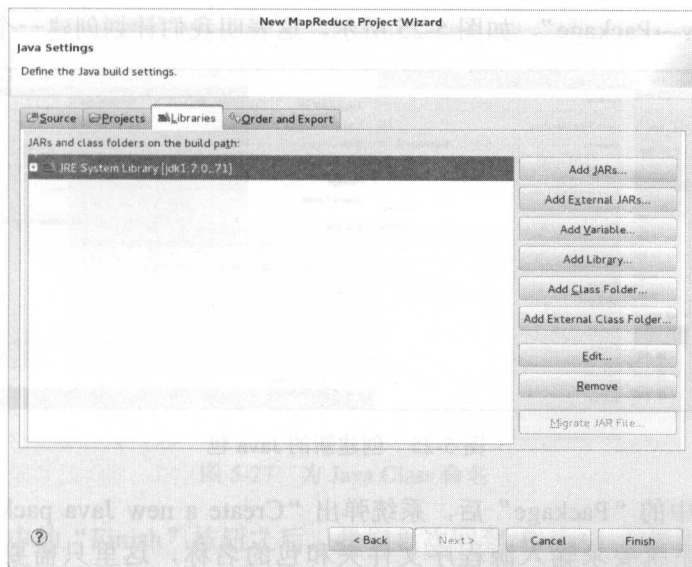


图 5-23 Eclipse 自动感知到系统安装的 Java 运行时环境 (JRE)

这说明，Eclipse 平台自动感知到了已经安装好的 JRE，因此，我们只要单击“Finish”按钮，返回主界面即可。

当然，如果没有显示 JRE，就需要选择右边的“Add Library...”按钮来手动添加。完成上述配置，我们会看到在“Project Explorer 中”，新增了一个名为 WordCount 的项目，如图 5-24 所示。

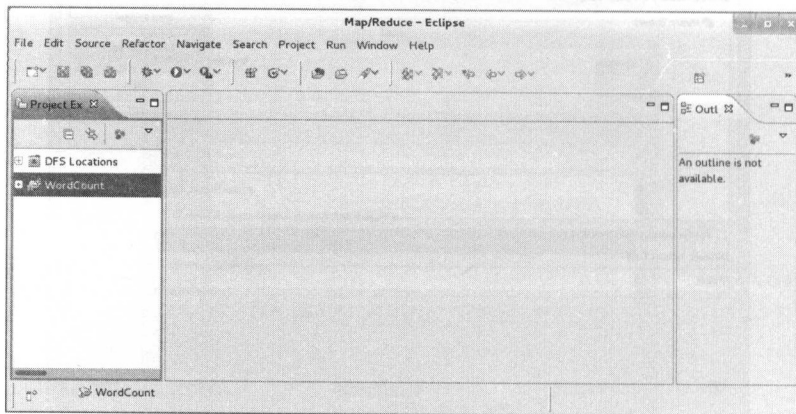


图 5-24 在 Eclipse 中新创建的 WordCount 项目

3. 编写源码

现在我们可以开始编写源程序了。首先选择 WordCount 项目下的“src”，单击鼠标右键，选择“New→Package”，如图 5-25 所示，这表明我们计划创建一个新的 Java 包。

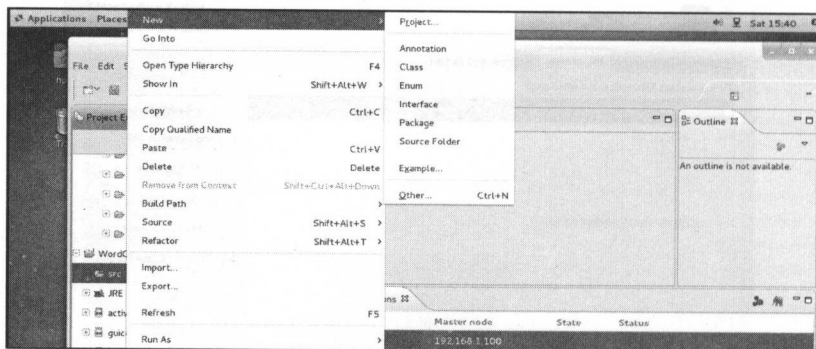


图 5-25 创建新的 Java 包

点选图 5-25 中的“Package”后，系统弹出“Create a new Java package”对话框，如图 5-26 所示。系统要求输入源程序文件夹和包的名称，这里只需要输入包名称，如 com.csu，单击“Finish”按钮即可。

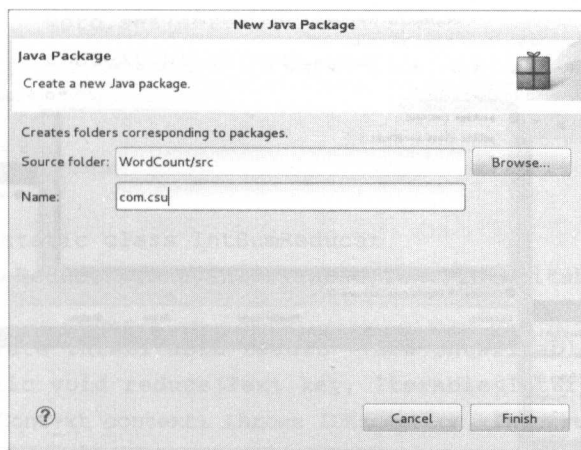


图 5-26 为 Java 包命名

单击图 5-26 中的“Finish”按钮之后，我们回到 Eclipse 主界面。接着用鼠标选择刚才创建的包 `com.csu`，单击鼠标右键，选择“New→Class”，在弹出的对话框中输入 Class 名，如 `WordCount`，如图 5-74 所示。



图 5-27 为 Java Class 命名

单击图 5-27 中的“Finish”按钮之后，我们再次回到 Eclipse 主界面，这时系统打开了类编辑窗口，如图 5-28 所示。

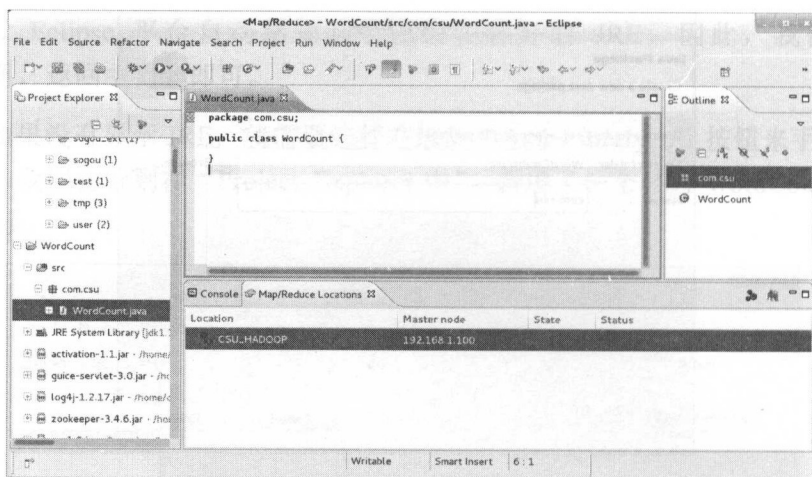


图 5-28 系统打开的 Java Class 编辑窗口

请读者将下面的代码复制到 WordCount.java 文件编辑区。

```
package com.csu;
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;
public class WordCount {
    public static class TokenizerMapper
    extends Mapper<Object, Text, Text, IntWritable>{
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();
        public void map(Object key, Text value, Context context
            ) throws IOException, InterruptedException
        {
            StringTokenizer itr = new StringTokenizer(value.toString());
            while (itr.hasMoreTokens()) {
```

```

        word.set(itr.nextToken());
        context.write(word, one);
    }
}

public static class IntSumReducer
extends Reducer<Text,IntWritable,Text,IntWritable>
{
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
        Context context) throws IOException, InterruptedException
    {
        int sum = 0;
        for (IntWritable val : values)
        {
            sum += val.get();
        }
        result.set(sum);
        context.write(key, result);
    }
}

public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf, args).
        getRemainingArgs();

    if (otherArgs.length < 2) {
        System.err.println("Usage: wordcount <in> [<in>...] <out>");
        System.exit(2);
    }

    Job job = new Job(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    for (int i = 0; i < otherArgs.length - 1; ++i) {
        FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
    }
}

```

```

        FileOutputFormat.setOutputPath(job,
            new Path(otherArgs[otherArgs.length - 1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}

```

实际上,上述代码是由 Hadoop 系统提供的示例程序,但是修改了一下包的名称,由原来的 `org.apache.hadoop.examples` 改为 `com.csu`。

这里顺便提示一下如何从 Hadoop 安装目录中获得上述源代码文件。

其实,在 Hadoop 的安装目录 “`hadoop-2.6.0/share/hadoop/MapReduce/sources/`” 下,有一个名为 `hadoop-MapReduce-examples-2.6.0-sources.jar` 的文件,用鼠标选择该文件,然后单击右键,在弹出的对话框中,选择 “Open With Archive Manager”,如图 5-29 所示,在打开 Jar 包后,深入到 `WordCount.java` 文件,双击即可用 `gedit` 打开该文件了。



图 5-29 用系统自带的 Archive Manager 打开 jar 包

关于 `WordCount` 程序,我们首先从总体上认识一下。

首先, `main` 函数调用 `Job` 类进行 `MapReduce` 作业的初始化:

```
Job job = new Job(conf, "word count");
```

早期的 MapReduce 会使用 Jobconf 类来对 MapReduce Job 进行初始化, 然后调用 `setJobName()` 方法命名这个 Job。对 Job 进行合理的命名有助于更快地找到 Job, 以便在 JobTracker 和 Tasktracker 的页面中对其进行监视。

```
JobConf conf = new JobConf(WordCount.class); conf.setJobName("word count");
```

接着设置 Job 输出结果<key,value>的中 key 和 value 数据类型。因为结果是<单词,个数>, 所以 key 设置为 Text 类型, 相当于 Java 中 String 类型; Value 设置为 IntWritable, 相当于 Java 中的 int 类型。

```
conf.setOutputKeyClass(Text.class);
conf.setOutputValueClass(IntWritable.class);
```

然后设置 Job 处理的 Map (拆分)、Combiner (中间结果合并) 及 Reduce (合并) 的相关处理类。这里用 Reduce 类来进行 Map 产生的中间结果合并, 避免给网络数据传输产生压力。

```
conf.setMapperClass(Map.class);
conf.setCombinerClass(Reduce.class);
conf.setReducerClass(Reduce.class);
```

接着调用 `addInputPath()` 和 `setOutputPath()` 设置输入输出路径。

```
FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
FileOutputFormat.setOutputPath(job, new Path(otherArgs[otherArgs.length - 1]));
```

以上就是 main 函数。

从整个应用程序看, 除了主函数以外, 还有两个子类, 分别是 `TokenizerMapper` 和 `IntSumReducer`, 这两个子类分别用来定义 Map 和 Reduce 函数。我们知道, MapReduce 正是通过为程序员提供这两个高度抽象的函数, 从而大大简化了分布式并行计算的设计过程。我们来看一下这两个函数的使用。

Map 方法分析:

```
public static class Map extends MapReduceBase implements
    Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();
    public void map(LongWritable key, Text value,
        OutputCollector<Text, IntWritable> output, Reporter reporter)
        throws IOException {
```



```

String line = value.toString();
StringTokenizer tokenizer = new StringTokenizer(line);
while (tokenizer.hasMoreTokens()) {
    word.set(tokenizer.nextToken());
    output.collect(word, one);
}
}
}

```

Map 类继承自 `MapReduceBase`，并且它实现了 `Mapper` 接口，此接口是一个规范类型，它有 4 种形式的参数，分别用来指定 Map 的输入 key 值类型、输入 value 值类型、输出 key 值类型和输出 value 值类型。在本例中，因为使用的是 `TextInputFormat`，它的输出 key 值是 `LongWritable` 类型，输出 value 值是 `Text` 类型。所以，Map 的输入类型为 `<LongWritable,Text>`。在本例中，我们需要输出 `<word,1>` 这样的形式，因此输出的 key 值类型是 `Text`，输出的 value 值类型是 `IntWritable`。

实现此接口类还需要实现 `Map` 方法，`Map` 方法会具体负责对输入进行操作，在本例中，`Map` 方法对输入的行以空格为单位进行切分，然后使用 `OutputCollect` 收集输出的 `<word,1>`。

Reduce 方法分析：

```

public static class Reduce extends MapReduceBase implements
    Reducer<Text, IntWritable, Text, IntWritable> {
    public void reduce(Text key, Iterator<IntWritable> values,
        OutputCollector<Text, IntWritable> output, Reporter reporter)
        throws IOException
    {
        int sum = 0;
        while (values.hasNext())
        {
            sum += values.next().get();
        }
        output.collect(key, new IntWritable(sum));
    }
}

```

Reduce 类也是继承自 `MapReduceBase` 的，需要实现 `Reducer` 接口。Reduce 类以 Map 的输出作为输入，因此 Reduce 的输入类型是 `<Text, IntWritable>`。而 Reduce 的输出是单词和它的数目，因此，它的输出类型是 `<Text,IntWritable>`。Reduce 类也要实现 `Reduce` 方

法,在此方法中,Reduce 函数将输入的 key 值作为输出的 key 值,然后将获得多个 value 值加起来,作为输出的值。

以上实际上是在讨论 Map 和 Reduce 函数的应用。如果想深入理解 Map 和 Reduce 函数的实现,则需要进一步阅读实现 Map 和 Reduce 函数的 Java 源代码(有关如何阅读 Hadoop 源代码的问题,读者可以参见 Hadoop 技术内幕一类的书籍或文献)。

下面我们来展示 Map 和 Reduce 函数中的关键概念。

(1) InputFormat 和 InputSplit。InputSplit 是 Hadoop 定义的用来传送给每个单独的 Map 的数据,InputSplit 存储的并非数据本身,而是一个分片长度和一个记录数据位置的数组。生成 InputSplit 的方法可以通过 InputFormat()来设置。

当数据传送给 Map 时,Map 会将输入分片传送到 InputFormat,InputFormat 则调用方法 getRecordReader()生成 RecordReader,RecordReader 再通过 creatKey()、creatValue()方法创建可供 Map 处理的<key,value>对。简而言之,InputFormat()方法是用来生成可供 Map 处理的<key,value>对的。

其中 TextInputFormat 是 Hadoop 默认的输入方法,在 TextInputFormat 中,每个文件(或其一部分)都会单独地作为 Map 的输入,而这个是继承自 FileInputFormat 的。之后,每行数据都会生成一条记录,每条记录则表示成<key,value>形式:

key 值是每个数据的记录在数据分片中字节偏移量,数据类型是 LongWritable; value 值是每行的内容,数据类型是 Text。

(2) OutputFormat。每一种输入格式都有一种输出格式与其对应。默认的输出格式是 TextOutputFormat,这种输出方式与输入类似,会将每条记录以一行的形式存入文本文件。不过,它的键和值可以是任意形式的,因为程序内部会调用 toString()方法将键和值转换为 String 类型再输出。

4. 运行程序

在运行程序之前,需要完成“Run Configuration”。将鼠标移动到 WordCount.java 代码编辑区,单击鼠标右键,在弹出的列表式菜单中,选择“Run As →Run Configurations”,如图 5-30 所示。

點選图 5-30 中的“Run Configurations”以后,系统将弹出配置对话框中。用户首先选择左边的 Java Application,然后单击鼠标右键,在弹出的列表中选择 New,表示新建一个配置,如图 5-31 所示。

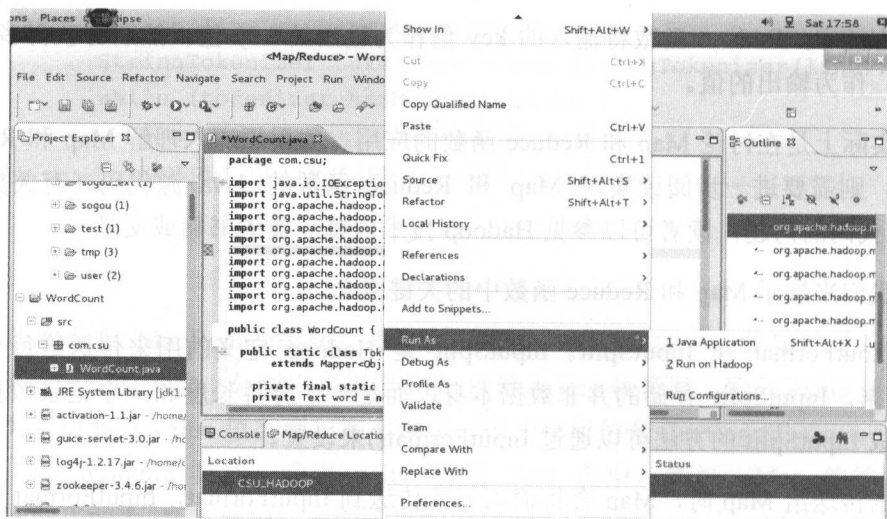


图 5-30 准备配置 Run Configurations

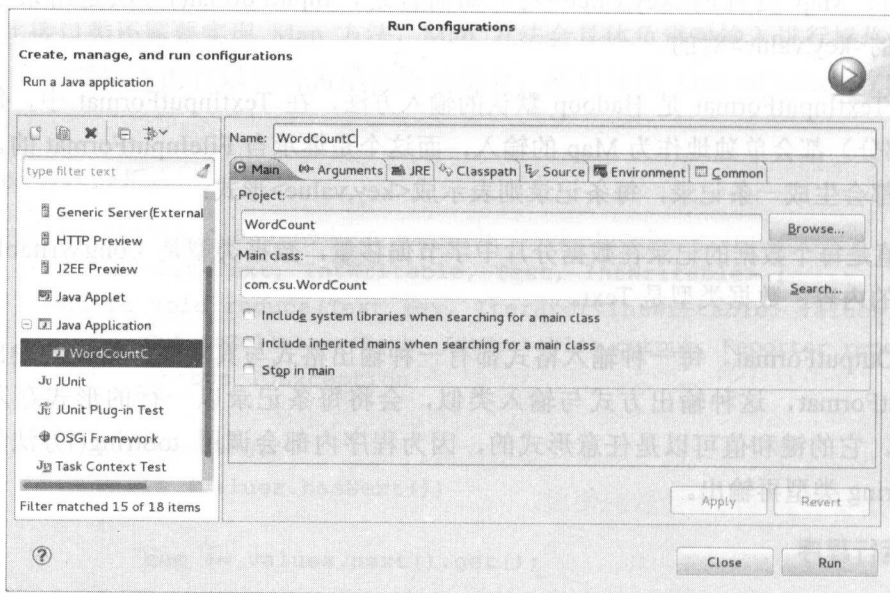


图 5-31 为 Java Application 创建一个新的配置

图 5-31 中，WordCountC 是用户给出的配置名称，可以任意命名，但不能和已经存在的配置同名。项目（Project）、主类名（Main Class）必须与用户创建项目的时候的设置一致，例如我们这里分别是 WordCount 和 com.csu.WordCount。接着打开“Arguments”标签页，以便配置运行参数，如图 5-32 所示。

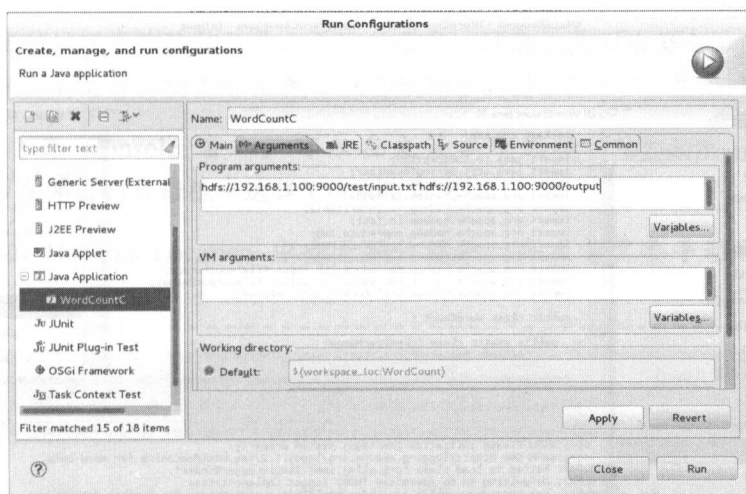


图 5-32 配置运行参数 (Arguments)

其中, `hdfs://192.168.1.100:9000/test/input.txt` 是输入文件, 也就是待处理的文件; 后面空一格后紧跟着的是输出目录, 即 `hdfs://192.168.1.100:9000/output`。

完成上述配置后, 请单击“Apply”按钮, 然后单击“Close”按钮。

回到主界面后, 将鼠标移动到代码编辑区, 单击右键, 选择“Run As”, 进一步选择“Run on Hadoop”, 系统弹出如图 5-33 所示的“Select Hadoop Location”对话框, 选择已经创建的一个 Hadoop Location, 单击“Finish”按钮即可。

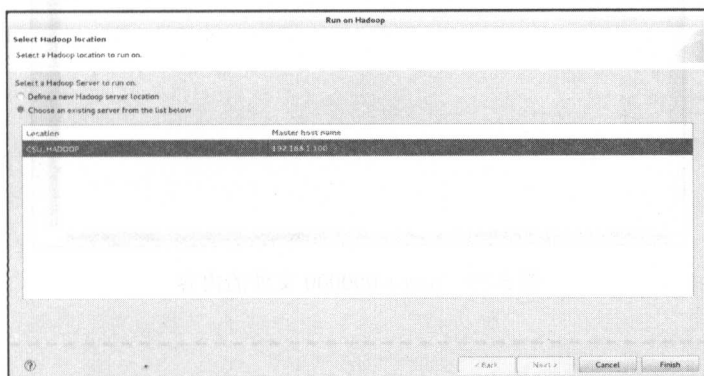


图 5-33 执行 Run on Hadoop 时选择一个 Location

图 5-34 是执行完毕后的主界面。我们看到 Hadoop 的 test 目录下新增了 output 子目录, 下面有两个文件, 其中 `part-r-000000` 包含了计算结果, 我们双击该文件即可看到里面的内容。

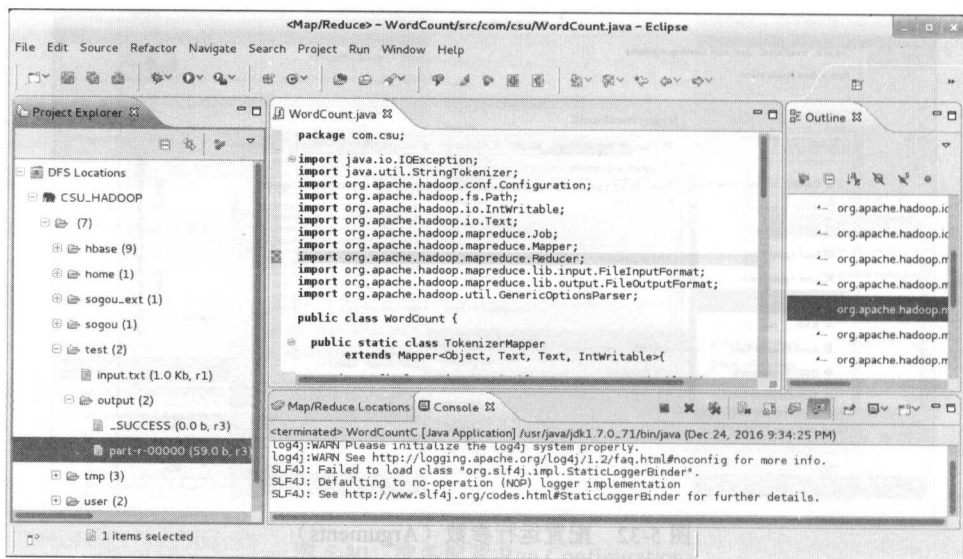


图 5-34 程序执行成功后系统在 Hadoop test 目录下生成了 output 子目录

图 5-35 显示了 part-r-000000 文件的内容。我们看到，处理结果与我们的预期完全一致。至此，我们就在 Eclipse 中完成了一个 Java 语言的 MapReduce 程序设计。

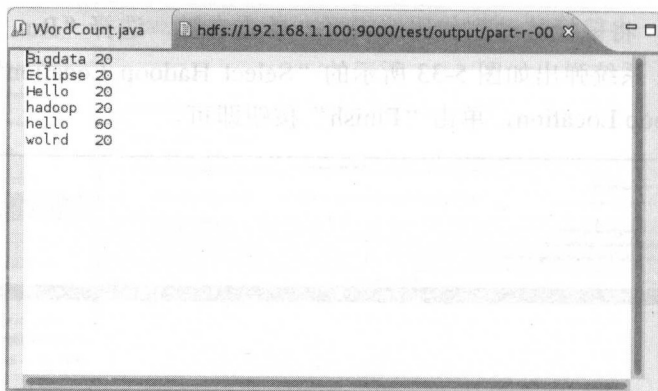


图 5-35 part-r-000000 文件的内容

注意事项

(1) 运行 MapReduce 程序时，Eclipse 会要求选择一个已经存在的配置，可以在弹出的对话框中单击选择一个自己做好的配置。如果在 Eclipse 中存在多个配置，要选择适合当前程序需要的配置（例如包含了特定程序运行参数的配置）。

(2) 再次运行程序时，Console 窗口可能会提示输出目录 output 已经存在，弹出

异常信息。这时候可先删除该目录，再运行程序。注意，是删除整个 output 目录；如果仅仅删除里面的文件，仍然会出现目录已经存在的提示。

(3) 要打开 Console 窗口，可以在主菜单中选择“Window→Show View→Other”，然后在弹出的对话框中选择 Console 即可。

(4) 要在 Hadoop Location 中查看操作结果，每次都需要手动刷新一下状态 (Refresh)。

5.5 本章小结

本章首先介绍了 MapReduce 的基本原理，在此基础上，描述了 MapReduce v2 也就是 Yarn 的架构和执行流程。这些知识是我们掌握 Hadoop 平台的重要基础。本章重点介绍了如何设计 MapReduce 程序，我们给出了一个在 Eclipse 中实现 Java 语言 MapReduce 程序的完整过程。

第 6 章

Hive 和 HBase 的安装与应用

本章主要介绍 Hive 和 HBase 的安装配置和基本应用方法，同时也介绍 MySQL 和 ZooKeeper 的安装与应用。需要说明的是，虽然 MySQL 是一个关系型数据库管理系统 (RDBMS)，但基于 Hadoop 的大数据应用也常常需要 MySQL 的支持，例如，Hive 就是依托 MySQL 存储其元数据的。正是由于这个原因，本章把 MySQL 的安装与基本应用也包含进来，况且本书后面的章节还将应用到 MySQL。ZooKeeper 是一个通用协调器，HBase 可以使用自带的 ZooKeeper，也可以使用独立安装的 ZooKeeper，因此在本章还介绍独立安装的 ZooKeeper。

6.1 在 CentOS7 下安装 MySQL

就数据库而言，CentOS6 或早期版本中提供的是 MySQL 的服务器/客户端安装包，但 CentOS7 则使用 MariaDB 替代了默认的 MySQL。MariaDB 数据库管理系统是 MySQL 的一个分支，主要由开源社区维护，采用 GPL 授权许可。MariaDB 完全兼容 MySQL，包括 API 和命令行。但是，本书暂不考虑从 MySQL 迁移到 MariaDB 的问题，我们这里继续提供在 CentOS7 下安装 MySQL 的指导，这是因为，MySQL 目前仍然是主流的开源数据库组件，同时熟悉 MySQL 也有利于读者全面了解相关技术。

6.1.1 下载或复制 MySQL 安装包

读者可以从 <http://www.MySQL.com/downloads/> 下载到最新版的 MySQL，也能够在本书第 6 章软件资源文件夹中，找到下列 MySQL 安装文件包，采用下面这些文件即可进行安装。

```
Mysql -community-client-5.7.13-1.el6.x86_64.rpm  
mysql -community-common-5.7.13-1.el6.x86_64.rpm  
mysql -community-libs-5.7.13-1.el6.x86_64.rpm  
mysql -community-server-5.7.13-1.el6.x86_64.rpm
```

首先, 请将这些文件复制到 master 的 /home/csu 目录下 (读者的目录可以不一样)。

注意, 安装和启动 MySQL 都需要使用 root 账户, 所以请首先输入 su root 命令并通过密码 (本书是 csucsu) 认证切换到 root 用户, 然后进入 /home/csu 目录。

6.1.2 执行安装命令

由于 CentOS7 已经存在 MariaDB 文件包, 因此在安装 MySQL 前需要先将其删除。读者在终端中输入命令 “rpm -qa | grep mariadb”, 可以检查现有的 MariaDB 安装包, 如图 6-1 所示。

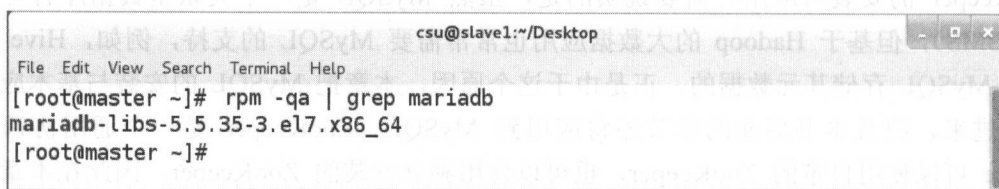


图 6-1 检查 CentOS7 已经存在的 MariaDB 文件包

我们看到, 系统中的 MariaDB 包是 mariadb-libs-5.5.35-3.el7.x86_64, 因此, 需要将其删除掉, 使用的命令是:

```
rpm -e --nodeps mariadb-libs-5.5.35-3.el7.x86_64
```

其中参数 nodeps 表示强制卸载, 即不考虑依赖项, 如图 6-2 所示。

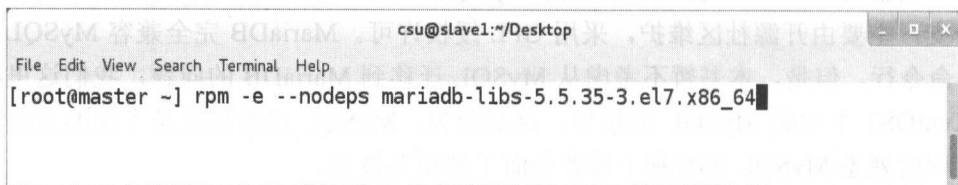


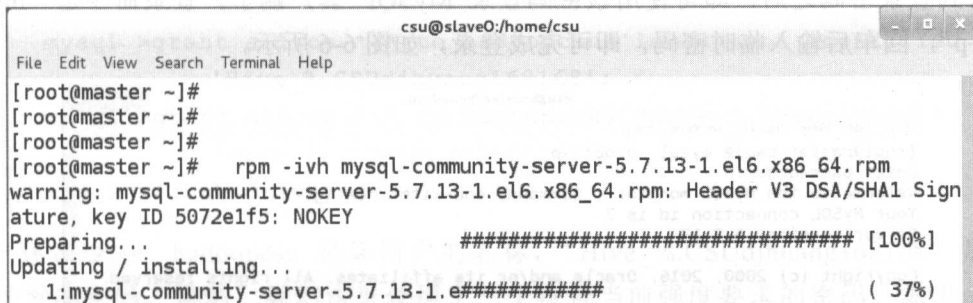
图 6-2 强制删除 MariaDB 文件包

删除完成后, 就可以开始安装 MySQL 了。在终端命令行分别执行以下四条命令, 依次安装 MySQL 组件。

```
rpm -ivh mysql -community-common-5.7.13-1.el6.x86_64.rpm
```

```
rpm -ivh mysql-community-libs-5.7.13-1.el6.x86_64.rpm
rpm -ivh mysql-community-client-5.7.13-1.el6.x86_64.rpm
rpm -ivh mysql-community-server-5.7.13-1.el6.x86_64.rpm
```

每执行一条命令，如果顺利，都将看到类似图 6-3 所示的打印信息。实际上，rpm 命令中，参数 i 表示安装，v 表示更多细节信息，h 表示显示进度信息。

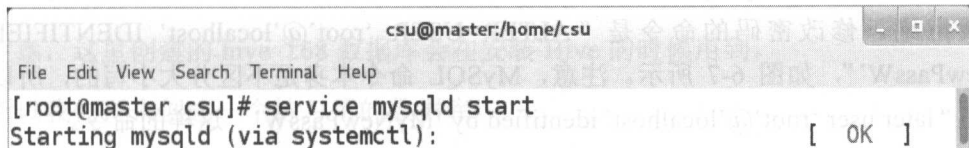


```
csu@slave0:/home/csu
File Edit View Search Terminal Help
[root@master ~]#
[root@master ~]#
[root@master ~]#
[root@master ~]# rpm -ivh mysql-community-server-5.7.13-1.el6.x86_64.rpm
warning: mysql-community-server-5.7.13-1.el6.x86_64.rpm: Header V3 DSA/SHA1 Signature, key ID 5072e1f5: NOKEY
Preparing... ##### [100%]
Updating / installing...
 1:mysql-community-server-5.7.13-1.el6.x86_64.rpm (37%)
```

图 6-3 安装 MySQL

6.1.3 启动 MySQL

执行上述安装命令后，即可立即启动 MySQL 服务了。命令是“service mysqld start”，如图 6-4 所示。



```
csu@master:/home/csu
File Edit View Search Terminal Help
[root@master csu]# service mysqld start
Starting mysqld (via systemctl): [ OK ]
```

图 6-4 启动 MySQL

启动后的 MySQL 作为一种服务在后台运行。

6.1.4 登录 MySQL

MySQL 启动以后，用户可以从终端登录系统。但是，首次登录需要获取自动生成的临时密码，命令是：

```
sudo grep 'temporary password' /var/log/MySQLd.log
```

回车后，即可显示自动生成的临时密码，本书是“ses%,9GwqiRQ”，如图 6-5 所示，读者的可能不一样。

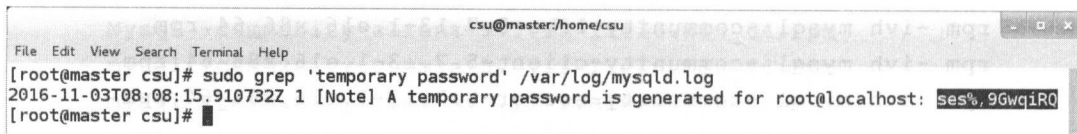


图 6-5 查看 MySQL root 用户的临时密码

得到登录密码之后，即可使用该密码登录 MySQL 客户端了。登录命令是“mysql -uroot -p”，回车后输入临时密码，即可完成登录，如图 6-6 所示。

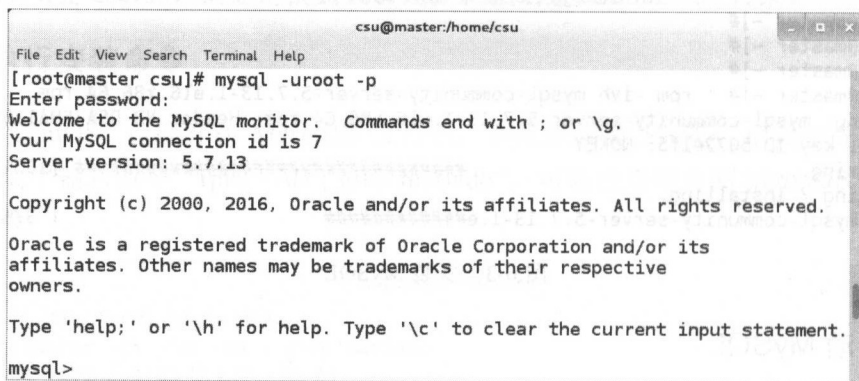


图 6-6 登录 MySQL

注意，首次登录 MySQL 后，需要修改临时密码才能开始创建数据库之类的工作，否则将被拒绝。修改密码的命令是“ALTER USER ‘root’@‘localhost’ IDENTIFIED BY ‘MyNewPassW’”，如图 6-7 所示。注意，MySQL 命令本身是不区分大小写的，所以也可以使用“later user ‘root’@‘localhost’ identified by ‘MyNewPassW’”这样的命令。



图 6-7 修改 MySQL root 用户密码

在修改用户密码的时候，有一个密码强度的问题。这一点也需要特别注意。像上面给出的“MyNewPassW”可能并不能满足系统当前设置的密码强度要求。这时系统会给出提示，需要采用强度更高的密码，一般要求含有大小写字母、符号（如%）和数字等，例如“qwerty,1289Qwdefi(AUL)”。建议读者设置好符合强度要求的密码之后，将该密码记录下来，以免遗忘造成麻烦。当然，为了学习方便，也可以修改 MySQL 的密码强度，通过将其设置为 Low，从而允许使用简单密码。我们这里不做这项操作，读者可以参考 MySQL 官方文档。

6.1.5 使用 MySQL

(1) 创建新用户。下面创建一个名称是 `hadoopcsu` 的 MySQL 新用户，需要连续执行以下四条命令：

```
mysql >grant all on *.* to hadoopcsu@'%' identified by
        'Hive_%,CSUdjhuang168168';
mysql >grant all on *.* to hadoopcsu@'localhost' identified by
        'Hive_%,CSUdjhuang168168';
mysql >grant all on *.* to hadoopcsu@'master' identified by
        'Hive_%,CSUdjhuang168168';
mysql >flush privileges;
```

上述命令中，`hadoopcsu` 是新用户的名称，“Hive_%,CSUdjhuang168168”是我们为其设置的密码。显然，我们这里使用了一个符合当前强度要求的密码。如果命令中设置的密码不符合强度要求，系统会拒绝创建，并给出提示信息。操作人员可以重新尝试创建。

上述练习不仅是为了熟悉 MySQL 命令，而且是为后面安装 Hive 进行必要的准备。

(2) 创建数据库。

```
mysql>create database hive_168;
```

注意，这里创建的 `hive_168` 数据库会在安装 Hive 的时候用到。

所以我们再来创建一个练习用的数据库。

```
mysql>create database test_db;
```

(3) 查看数据库。

```
mysql >show databases;
```

(4) 使用数据库。

```
mysql >use test_db;
```

(5) 创建表。

```
mysql >create table myclass(
        >id int(4) not null primary key auto_increment,
        > name char(20) not null,
        > sex int(4) not null default '0',
        > degree double(16,2));
```

(6) 插入数据。

```
mysql >insert into myclass values (0001,"Huang Dong Jun",0,100.00);
```

(7) 查询数据。

```
mysql >select * from myclass;
```

(8) 删除表。

```
mysql>drop table myclass;
```

(9) 删除数据库。

```
mysql>drop database test_db;
```

(10) 退出 MySQL。

```
mysql>quit;
```

6.1.6 问题与解决办法

有很多用户在安装 MySQL 后不能启动系统服务，或者首次启动成功后下次再启动的时候失败，可能看到如图 6-8 所示的提示信息。

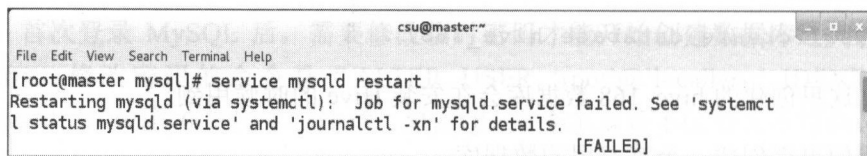


图 6-8 启动 MySQL 失败

为了解决这个问题，我们不妨查看一下日志文件，“cat /var/log/mysqld.log”，如图 6-9 所示。

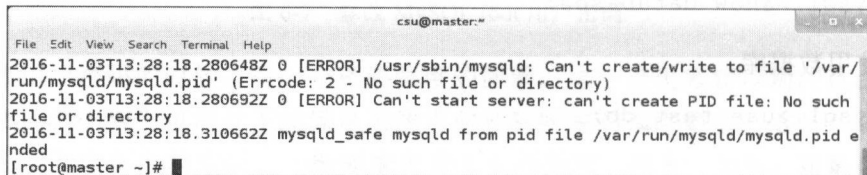


图 6-9 cat /var/log/mysqld.log 显示的部分信息

从显示信息中，我们发现是无法创建 PID 文件（can't create PID file: No such file or directory），提示没有该文件或目录。用 ls 命令查看一下，果然“/var/run”下没有 mysqld 目录。因此，我们可以创建该目录，命令是“mkdir -p /var/run/mysqld”，如图 6-10 所示。

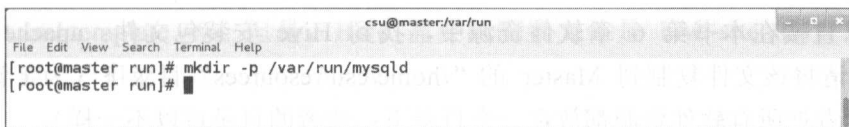


图 6-10 创建 MySQLd 目录

再次执行“service mysqld start”，仍然不能启动 MySQL。是什么原因呢？原来刚才创建的 mysqld 目录的属主和属组仍然是 root，而 MySQL 不能在该权限的目录中创建文件。因此，需要修改“/var/run/mysqld/”的属主和属组，命令是“chown mysql.mysql /var/run/mysqld/”，修改以后就可以正常启动 MySQL 了，如图 6-11 所示。

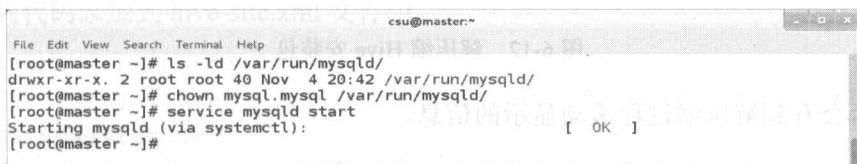


图 6-11 修改 mysqld 目录属性

令人遗憾的是，上面创建的 mysqld 目录在关机以后会消失，下次启动 MySQL 需要再次创建。有研究人员认为这是 MySQL 的一个不合理之处。不过我们可以把上述两条命令写进 Linux 系统启动脚本中，问题也容易解决。

6.2 Hive 安装与应用

Hive 需要安装在成功部署的 Hadoop 平台上，并且要求 Hadoop 已经正常启动。所以，读者需要首先验证自己计算机上的 Hadoop 是否处于正常运行状态，方法是执行一个 Hadoop 命令，如“hdfs dfs -ls /”，看是否正常显示 HDFS 上的目录列表。同时，通过浏览器查看系统状态，地址是 <http://master:50070> 和 <http://master:18088>，查看结果应当与安装部分看到的情况一致。如果满足上述两个条件，说明 Hadoop 已经正常启动。

准备就绪后，就可以开始安装 Hive 了。我们打算将 Hive 安装在 Master 节点上，因此以下的操作均在 Master 节点上进行。同时，所有操作都使用 csu 用户，所以需要确保已经切换到 csu 用户。

6.2.1 下载并解压 Hive 安装包

读者可以从 <http://apache.fayea.com/hive/hive-2.1.0/> 下载 Hive-2.1.0 安装包。

也可以直接在本书第 6 章软件资源中,找到 Hive 安装包文件:apache-hive-2.1.0-bin.tar.gz。请将该文件复制到 Master 的“/home/csu/resources”目录下(为了管理方便,我们建议读者把所有软件资源都放在一个目录下;读者的目录可以不一样)。

首先,将 apache-hive-2.1.0-bin.tar.gz 文件从 resources 目录下再复制至“/home/csu/”,接着进入“/home/csu/”子目录,并执行解压 Hive 安装包的命令,如图 6-12 所示。

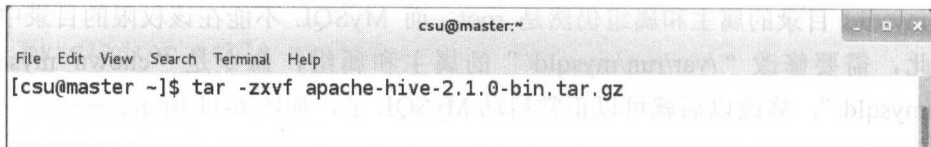


图 6-12 解压缩 Hive 安装包

回车后会看到解压缩过程滚动显示的信息。

执行完毕,我们可以查看一下安装情况。请切换到 apache-hive-2.1.0-bin 目录,执行命令“ls -l”,会看到如图 6-13 所示的内容,这些就是 Hive 系统文件和目录,说明解压缩成功。

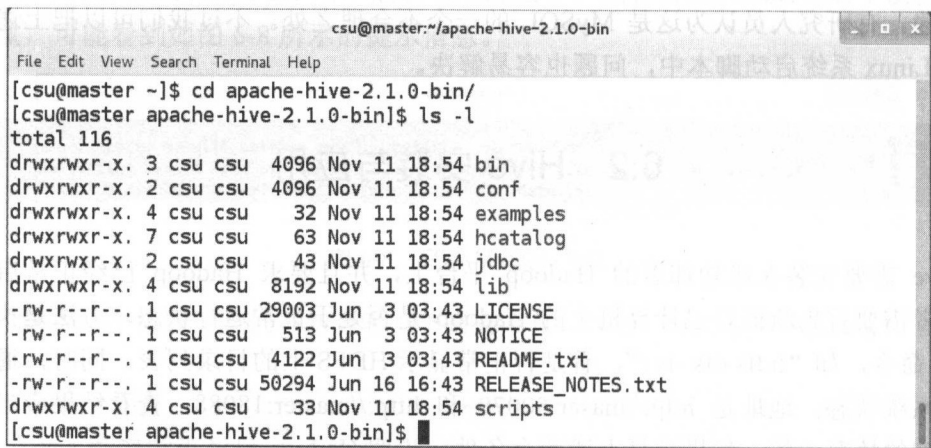


图 6-13 Hive 主安装目录中的内容

6.2.2 配置 Hive

完成上述解压缩之后,需要进行相关文件的创建和配置。

1. 创建 hive-site.xml 文件

实际上,在 Hive 安装目录下的配置目录 conf 中,系统给出了一些配置文件模板,如

hive-default.xml.template 等，但是，Hive 需要的配置文件是 hive-site.xml，而它并不存在，所以需要用户自己创建。

进入 conf 目录，执行“gedit hive-site.xml”命令，如图 6-14 所示，开始编辑 hive-site.xml 文件。

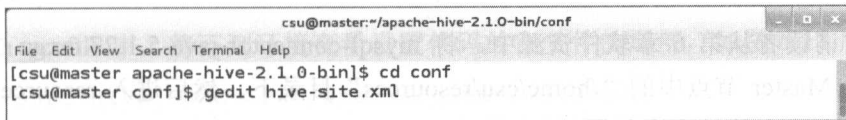


图 6-14 准备编辑 hive-site.xml 文件

将下列代码添加到 hive-site.xml 文件中。

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
  <property>
    <name>hive.metastore.local</name>
    <value>true</value>
  </property>
  <property>
    <name>javax.jdo.option.ConnectionURL</name>
    <value>jdbc:mysql://master:3306/hive_168?characterEncoding=
      UTF-8</value>
  </property>
  <property>
    <name>javax.jdo.option.ConnectionDriverName</name>
    <value>com.mysql.jdbc.Driver</value>
  </property>
  <property>
    <name>javax.jdo.option.ConnectionUserName</name>
    <value>hadoopcsu</value>
  </property>
  <property>
    <name>javax.jdo.option.ConnectionPassword</name>
    <value>Hive_%,CSUdjhuang168168</value>
  </property>
</configuration>
```

编辑完成，保存退出即可。用“ls -l”命令可以看到 conf 下增加了 hive-site.xml 文件。

注意，上述代码中，hive_168 正是我们前面在 MySQL 中创建的数据库，hadoopcsu 是我们前面创建的 MySQL 新用户，而 “Hive_%,CSUdjhuang168168” 则是在 MySQL 中创建 hadoopcsu 用户时所设置的密码。

2. 复制 java connector 到依赖库

接着，请读者从第 6 章软件资源中，将 mysql-connector-java-5.1.27.tar.gz 文件复制到自己计算机 Master 节点中的 “/home/csu/resources” 目录下，然后进入 resources 目录，在其中解压缩上述文件，命令如图 6-15 所示。

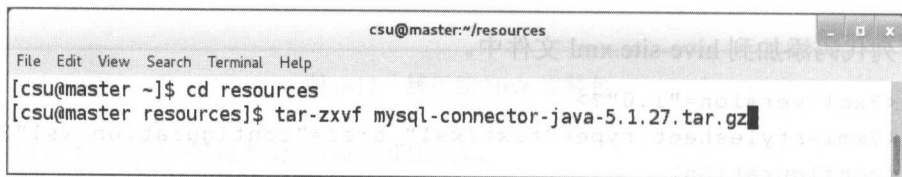


图 6-15 解压缩 Java 连接器安装包

回车执行命令后，系统将在 resources 下创建 mysql-connector-java-5.1.27 目录。进入该目录，然后将其中的 mysql-connector-java-5.1.27-bin.jar 文件复制到 Hive 的安装目录的依赖库目录下，即执行 “cp mysql-connector-java-5.1.27-bin.jar ~/apache-hive-2.1.0-bin/lib/” 命令，如图 6-16 所示。

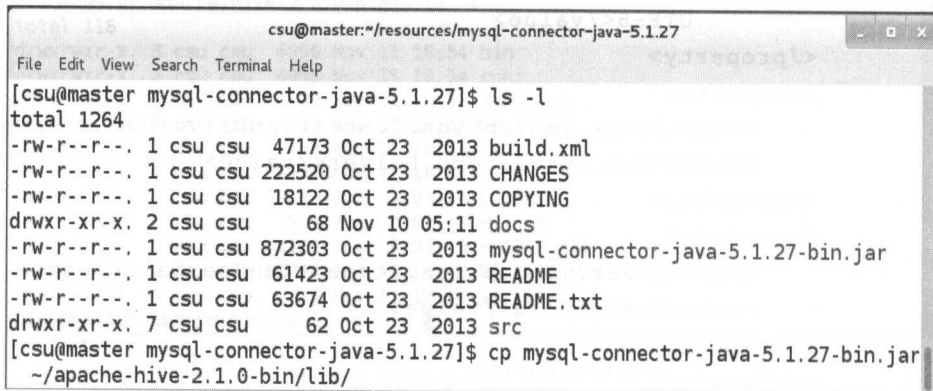


图 6-16 将 Java 连接器 jar 包放进 Hive 的 lib 目录下

3. 配置 .bash_profile 文件

我们知道，.bash_profile 文件是一个用户（如 csu）使用 Linux 的系统配置文件，所以自然也需要为 Hive 进行必要的配置。执行 “gedit /home/csu/.bash_profile” 命令，如图 6-17 所示。

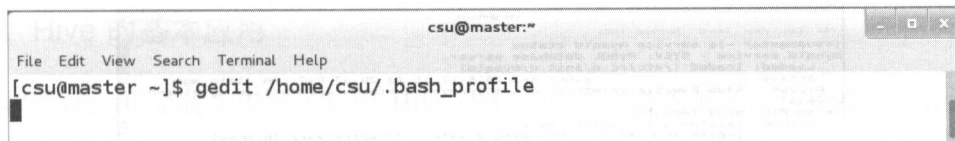


图 6-17 准备修改 Linux 用户的系统配置文件

在 `.bash_profile` 中，将下列环境变量设置代码放在文件的尾部，如图 6-18 所示。

```
export HIVE_HOME=$PWD/apache-hive-2.1.0-bin
export PATH=$PATH:$HIVE_HOME/bin
```

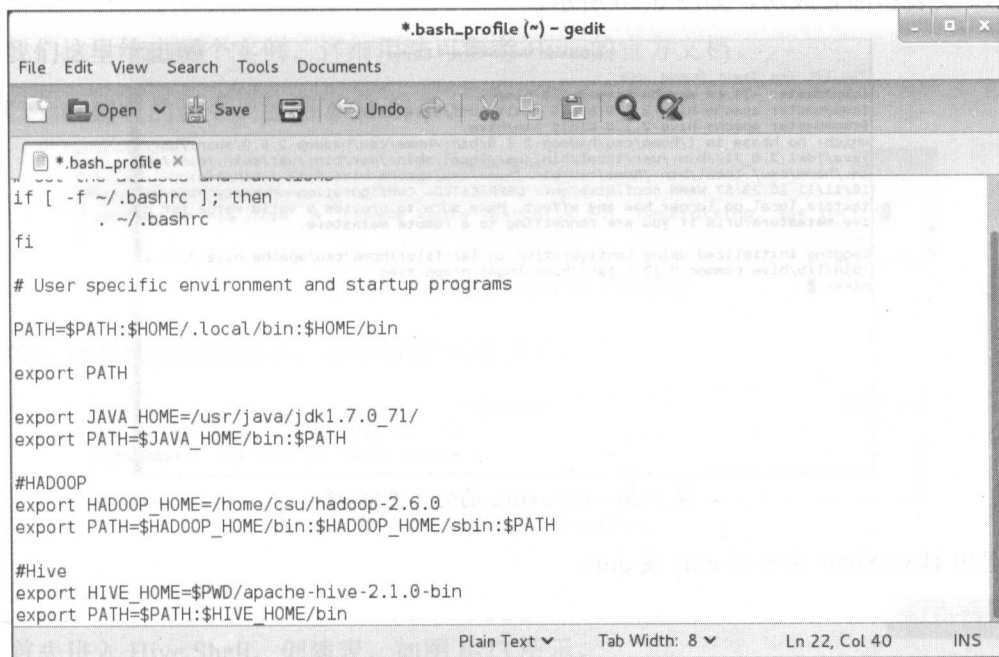


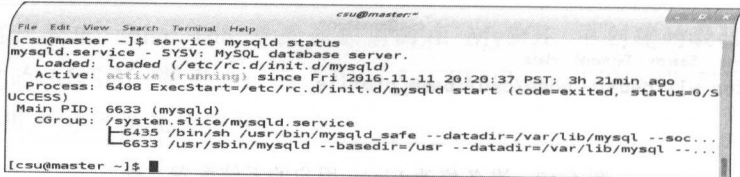
图 6-18 编辑系统配置文件

编辑完毕，保存文件并退出 `gedit` 即可。

至此，我们就完成了在 Master 上安装和配置 Hive。

6.2.3 启动并验证 Hive

要启动 Hive，必须保证 MySQL 服务已经启动。我们可以用 `service mysqld status` 命令查看 MySQL 服务的状态，如图 6-19 所示；如果提示信息中含有绿色的“active (running)”，表明 MySQL 服务处于启动状态。



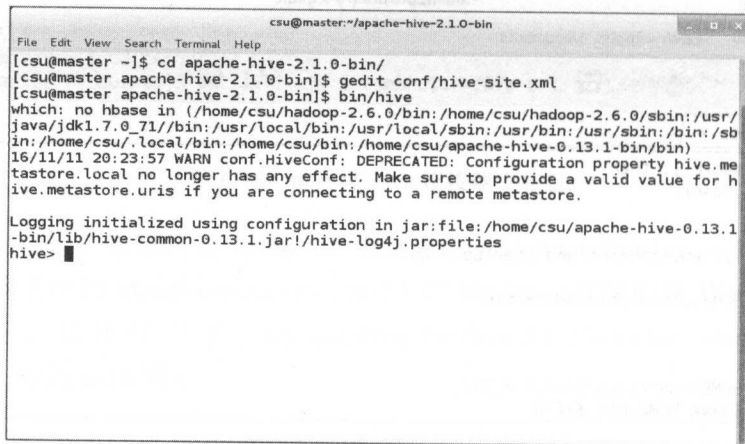
```

csu@master:~
[csu@master ~]$ service mysqld status
mysqld.service - SYSV: MySQL database server.
   Loaded: loaded (/etc/rc.d/init.d/mysqld)
   Active: active (running) since Fri 2016-11-11 20:20:37 PST; 3h 21min ago
     Process: 6408 ExecStart=/etc/rc.d/init.d/mysqld start (code=exited, status=0/SUCCESS)
    Main PID: 6633 (mysqld)
      CGroup: /system.slice/mysqld.service
              └─6435 /bin/sh usr/bin/mysqld_safe --datadir=/var/lib/mysql --soc...
                └─6633 /usr/sbin/mysqld --basedir=/usr --datadir=/var/lib/mysql --...
[csu@master ~]$

```

图 6-19 查看 MySQL 是否处于运行状态

在 MySQL 已经启动的条件下, 进入 Hive 安装目录, 执行 “bin/hive” 命令, 如果出现 Hive 命令提示符 “hive>”, 这是 Hive 交互式命令行 (即 Hive Shell 命令) 提示符, 表明 Hive 安装和部署成功, 如图 6-20 所示。



```

csu@master:~/apache-hive-2.1.0-bin
[csu@master ~]$ cd apache-hive-2.1.0-bin/
[csu@master apache-hive-2.1.0-bin]$ gedit conf/hive-site.xml
[csu@master apache-hive-2.1.0-bin]$ bin/hive
which: no hbase in (/home/csu/hadoop-2.6.0/bin:/home/csu/hadoop-2.6.0/sbin:/usr/java/jdk1.7.0_71/bin:/usr/local/bin:/usr/local/sbin:/usr/bin:/usr/sbin:/bin:/sbin:/home/csu/.local/bin:/home/csu/bin:/home/csu/apache-hive-0.13.1-bin/bin)
16/11/11 20:23:57 WARN conf.HiveConf: DEPRECATED: Configuration property hive.metastore.local no longer has any effect. Make sure to provide a valid value for hive.metastore.uris if you are connecting to a remote metastore.

Logging initialized using configuration in jar:file:/home/csu/apache-hive-0.13.1-bin/lib/hive-common-0.13.1.jar!/hive-log4j.properties
hive>

```

图 6-20 进入 Hive 的交互式命令行

退出 Hive Shell 命令是 exit 或 quit。

注意事项

在使用 Hive 的时候, 有些用户可能会遇到不能启动的问题。

例如, 输入 bin/hive 命令回车后, 出现如下的报错信息:

```
Exception in thread "main" java.lang.RuntimeException: Hive metastore
database is not initialized. Please use schematool (e.g. ./schematool -
initSchema -dbType ...) to create the schema. If needed, don't forget to
include the option to auto-create the underlying database in your JDBC
connection string (e.g. ?createDatabaseIfNotExist=true for mysql)
```

从给出的信息来看, 是因为元数据库没有初始化, 因此, 在 Linux 终端执行如下
的命令即可解决该问题:

```
schematool -dbType mysql -initSchema
```


6.2.4 Hive 的基本应用

本书的第 10 章将集中展示利用 Hive 进行交互式大数据处理。这里我们先介绍 Hive 的基本应用。

Hive 的应用有两种模式，即命令行模式和交互模式。命令行模式用起来如同“Hadoop fs -ls /”这样的方式，是在 Linux 提示符下进行的操作；交互模式则需要进入 Hive shell。注意，Hive Shell 里的每条命令后都要用分号结束。

1. 命令行模式

我们这里给出两个实例，详细用法可参考 Hive 的官方文档。

(1) 创建表的命令，如图 6-21 所示。

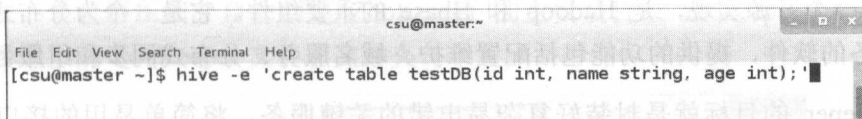


图 6-21 在 Linux 提示符下创建表

(2) 查看已经创建的表，命令如图 6-22 所示。

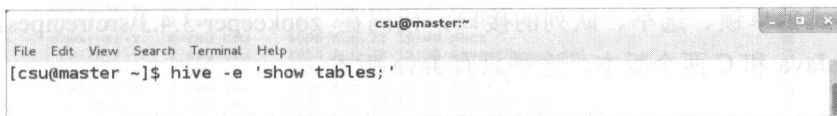


图 6-22 查看已经创建的表

2. Hive Shell 模式

首先进入 Hive Shell，创建表，如图 6-23 所示。

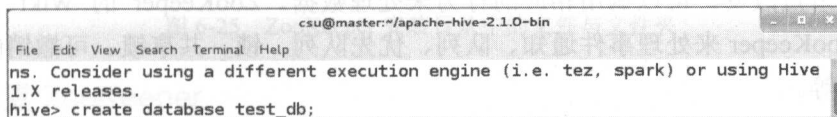


图 6-23 在 Hive Shell 中创建数据库

查看创建的数据库，命令如图 6-24 所示。

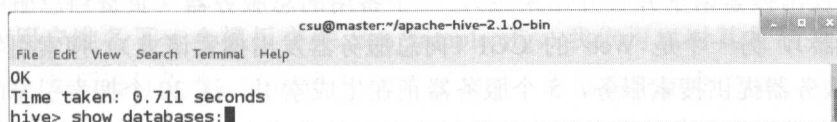


图 6-24 在 Hive Shell 中创建数据库

从上述示例大家可以发现，Hive 的命令行模式与 Shell 模式本质上是一致的，都是基于相同的计算引擎。

6.3 ZooKeeper 集群安装

6.3.1 ZooKeeper 简介

ZooKeeper 是 Hadoop Ecosystem 中非常重要的组件，它的主要功能是为分布式系统提供一致性协调（Coordination）服务，与之对应的 Google 的类似服务叫做 Chubby。

ZooKeeper 是一个分布式的，开放源码的分布式应用程序协调服务，是 Google 的 Chubby 的一个开源实现，是 Hadoop 和 Hbase 的重要组件。它是一个为分布式应用提供一致性服务的软件，提供的功能包括配置维护、域名服务、分布式同步和组服务等。

ZooKeeper 的目标就是封装好复杂易出错的关键服务，将简单易用的接口和性能高效、功能稳定的系统提供给用户。

ZooKeeper 包含一个简单的原语集，提供 Java 和 C 的接口。ZooKeeper 代码版本中，提供了分布式独享锁、选举、队列的接口，代码在 `zookeeper-3.4.3/src/recipes`，其中分布式锁和队列有 Java 和 C 两个版本，选举只有 Java 版本。

在 ZooKeeper 中，`znode` 是一个跟 UNIX 文件系统路径相似的节点，可以往这个节点存储或获取数据。如果在创建 `znode` 时 `Flag` 设置为 `EPHEMERAL`，那么当创建这个 `znode` 的节点和 ZooKeeper 失去连接后，这个 `znode` 将不再存在在 ZooKeeper 里，ZooKeeper 使用 `Watcher` 察觉事件信息。当客户端接收到事件信息，如连接超时、节点数据改变、子节点改变，可以调用相应的行为来处理数据。ZooKeeper 的 Wiki 页面展示了如何使用 ZooKeeper 来处理事件通知、队列、优先队列、锁、共享锁、可撤销的共享锁和两阶段提交等。

关于 ZooKeeper 的作用，我们来看一个简单的例子：假设我们有 20 个搜索引擎的服务器（每个负责总索引中的一部分的搜索任务）和一个总服务器（负责向这 20 个搜索引擎的服务器发出搜索请求合并结果集），一个备用的总服务器（负责当总服务器宕机时替换总服务器），另一个是 Web 的 CGI（向总服务器发出搜索请求）。搜索引擎的服务器中的 15 个服务器提供搜索服务，5 个服务器正在生成索引。这 20 个搜索引擎的服务器经常要让正在提供搜索服务的服务器停止提供服务开始生成索引，或生成索引的服务器已经

把索引生成完成，可以提供搜索服务了。使用 ZooKeeper 可以保证总服务器自动感知有多少提供搜索引擎的服务器并向这些服务器发出搜索请求，当总服务器宕机时自动启用备用的总服务器。

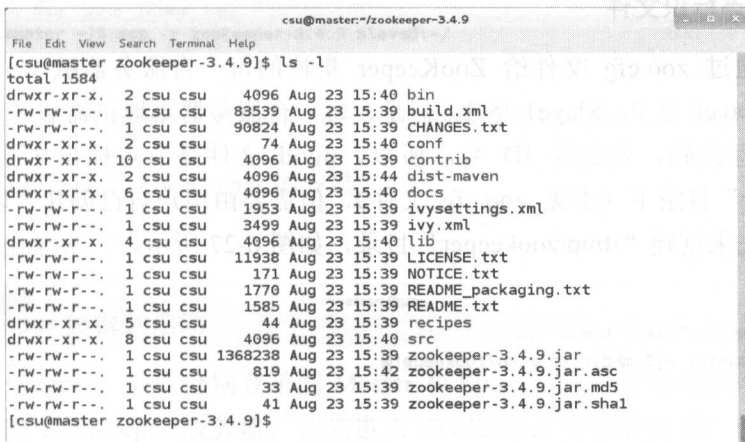
6.3.2 安装 ZooKeeper

读者可以从 <http://www.apache.org/dyn/closer.cgi/zookeeper/> 下载 ZooKeeper 安装包，也可以在本章软件资源文件夹中找到 zookeeper-3.4.9.tar.gz 文件。

请将上述文件复制到 Master 节点的 “/home/csu/” 目录下，进入该目录，然后执行解压缩命令：

```
tar -zxvf zookeeper-3.4.9.tar.gz
```

回车后，系统解压缩并自动创建 ZooKeeper 的主安装目录 zookeeper-3.4.9，图 6-25 给出了该目录下的 ZooKeeper 文件和文件夹。



```

csu@master:~/zookeeper-3.4.9
File Edit View Search Terminal Help
[csu@master zookeeper-3.4.9]$ ls -l
total 1584
drwxr-xr-x. 2 csu csu 4096 Aug 23 15:40 bin
-rw-rw-r--. 1 csu csu 83539 Aug 23 15:39 build.xml
-rw-rw-r--. 1 csu csu 90824 Aug 23 15:39 CHANGES.txt
drwxr-xr-x. 2 csu csu 74 Aug 23 15:40 conf
drwxr-xr-x. 10 csu csu 4096 Aug 23 15:39 contrib
drwxr-xr-x. 2 csu csu 4096 Aug 23 15:44 dist-maven
drwxr-xr-x. 6 csu csu 4096 Aug 23 15:40 docs
-rw-rw-r--. 1 csu csu 1953 Aug 23 15:39 ivysettings.xml
-rw-rw-r--. 1 csu csu 3499 Aug 23 15:39 ivy.xml
drwxr-xr-x. 4 csu csu 4096 Aug 23 15:40 lib
-rw-rw-r--. 1 csu csu 11938 Aug 23 15:39 LICENSE.txt
-rw-rw-r--. 1 csu csu 171 Aug 23 15:39 NOTICE.txt
-rw-rw-r--. 1 csu csu 1770 Aug 23 15:39 README_packaging.txt
-rw-rw-r--. 1 csu csu 1585 Aug 23 15:39 README.txt
drwxr-xr-x. 5 csu csu 44 Aug 23 15:39 recipes
drwxr-xr-x. 8 csu csu 4096 Aug 23 15:40 src
-rw-rw-r--. 1 csu csu 1368238 Aug 23 15:39 zookeeper-3.4.9.jar
-rw-rw-r--. 1 csu csu 819 Aug 23 15:42 zookeeper-3.4.9.jar.asc
-rw-rw-r--. 1 csu csu 33 Aug 23 15:39 zookeeper-3.4.9.jar.md5
-rw-rw-r--. 1 csu csu 41 Aug 23 15:39 zookeeper-3.4.9.jar.sha1
[csu@master zookeeper-3.4.9]$

```

图 6-25 ZooKeeper 安装目录中的文件与文件夹

6.3.3 配置 ZooKeeper

1. 服务器集群核心属性

ZooKeeper 的核心服务器属性配置文件是 zoo.cfg。在主安装目录下的 conf 子目录内，系统为用户准备了一个模板文件 zoo_sample.cfg，我们可以对其复制并改名，得到 zoo.cfg 文件，然后进行修改。首先进入 conf 子目录，然后执行命令 “cp zoo_sample.cfg zoo.cfg”，如图 6-26 所示。

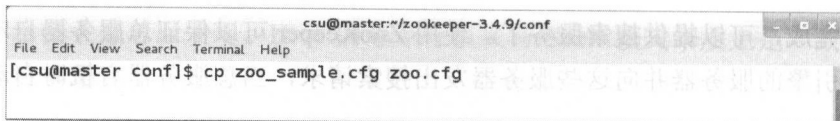


图 6-26 复制并重新命名 zoo.cfg 文件

接着用 gedit 编辑 zoo.cfg 文件。请将如下代码添加到文件的尾部。

```
server.1=master:2888:3888
server.2=slave0:2888:3888
server.3=slave1:2888:3888
```

上述代码是按照“服务器编号、服务器地址、LF 通信端口和选举端口”的顺序排列的，其中，server 表示 ZooKeeper 集群的服务器，我们这里配置了三台服务器，server 后面的数字代表服务器的 ID。等号后面紧跟的是服务器地址，我们这里使用了主机名。2888 是 LF（即 Leader 与 Follower 之间的）通信端口，而 3888 则是选举端口。

2. 创建节点标识文件

上面我们通过 zoo.cfg 文件给 ZooKeeper 集群的每一台服务器赋予了一个 ID 号，Master 是 1，Slave0 是 2，Slave1 是 3。但是，每一台服务器本地也需要一个 myid 文件，里面仅包含一行代码，就是其 ID 号。所以，myid 文件是节点标识文件，缺省放置在“/tmp/zookeeper”目录下（参见 zoo.cfg 文件），但需要由用户自己创建。以 Master 节点为例，我们首先来创建“/tmp/zookeeper”目录，如图 6-27 所示。

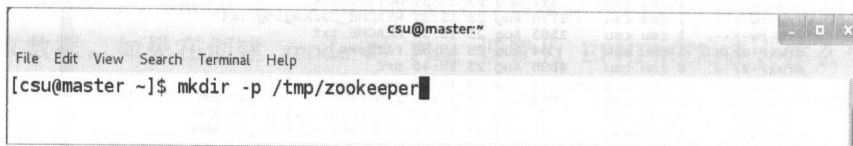


图 6-27 创建/tmp/zookeeper 目录

接着执行“gedit /tmp/zookeeper/myid”命令，如图 6-28 所示。

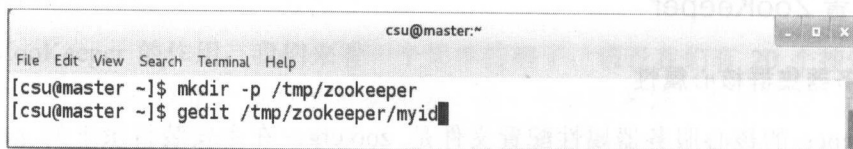


图 6-28 创建 myid 文件

然后在新建的 myid 文件中输入对应的 ID 号，保存退出即可，如图 6-29 所示。

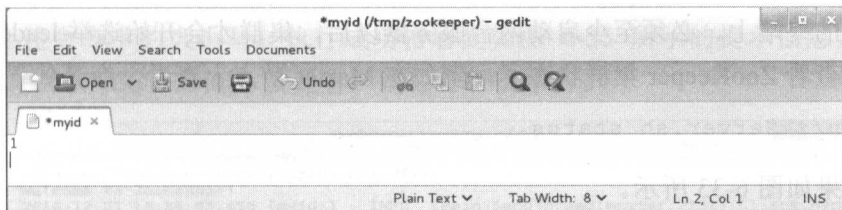


图 6-29 myid 文件仅仅包含一行

注意，其他 Slave 节点上也要进行同样的设置，只是 ID 号不同而已。

3. 复制 ZooKeeper 安装文件

最后，用户需要根据自己 Linux 集群的具体情况，将上面安装好的 ZooKeeper 文件复制到 Slave 节点。我们这里需要复制两次，分别是 Slave0 和 Slave1，其中，复制到 Slave0 的命令是“`scp -r zookeeper-3.4.9 slave0:~/`”，如图 6-30 所示。

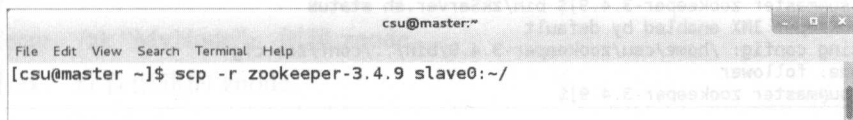


图 6-30 将 master 上的 ZooKeeper 安装目录复制到 slave0 的/home/csu/下

至此，我们就完成了 ZooKeeper 的安装。

6.3.4 启动和测试

1. ZooKeeper 集群的启动

要启动 ZooKeeper，需要分别登录到 Master 和 Slave 节点进行启动操作。例如，要启动 Master 节点的 ZooKeeper 服务器，首先进入 ZooKeeper 主安装目录，然后执行启动命令，如图 6-31 所示。请特别注意 `zkServer.sh` 中的大写字母，如果误写成小写的 `s`，系统会提示无该文件或目录（no such file or directory）。

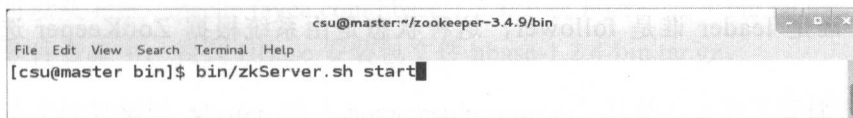


图 6-31 启动 Master 节点的 ZooKeeper 服务器

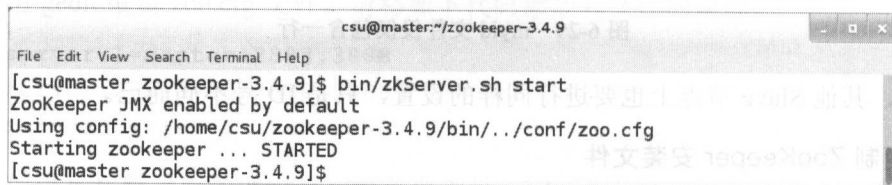
回车后，如果启动正常，将看到如图 6-32 所示的打印信息。

注意，我们还需要启动其他节点上的 ZooKeeper 服务器，启动命令一样，只是要切换

到这些节点的终端上。必须至少启动两台服务器以后，集群才会开始选举 leader 节点。这时，就可以查看 ZooKeeper 集群状态了，例如在 Master 节点上查看，命令如下：

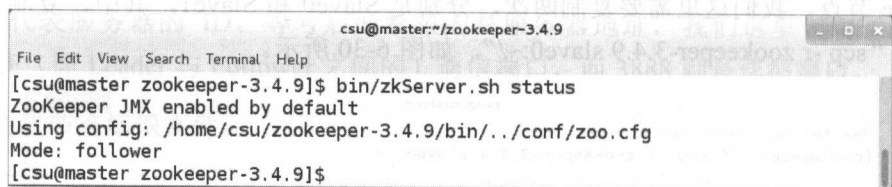
```
bin/zkServer.sh status
```

执行结果如图 6-33 所示。



```
csu@master:~/zookeeper-3.4.9
File Edit View Search Terminal Help
[csu@master zookeeper-3.4.9]$ bin/zkServer.sh start
ZooKeeper JMX enabled by default
Using config: /home/csu/zookeeper-3.4.9/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
[csu@master zookeeper-3.4.9]$
```

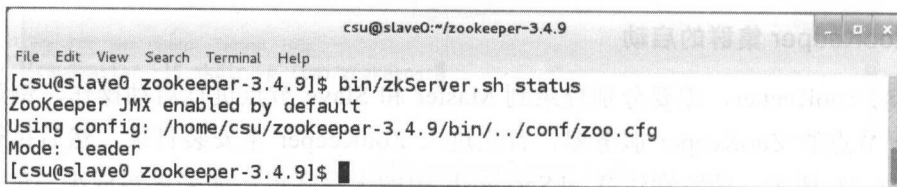
图 6-32 启动成功的显示



```
csu@master:~/zookeeper-3.4.9
File Edit View Search Terminal Help
[csu@master zookeeper-3.4.9]$ bin/zkServer.sh status
ZooKeeper JMX enabled by default
Using config: /home/csu/zookeeper-3.4.9/bin/../conf/zoo.cfg
Mode: follower
[csu@master zookeeper-3.4.9]$
```

图 6-33 查看 Master 节点上的 ZooKeeper 服务器状态

我们看到，系统给出的信息表明，该节点（Master）是一个 follower。根据 ZooKeeper 工作原理，集群中应当有一个 leader，不妨查看一下其他节点的状态，我们这里看到 Slave0 是 leader 节点，结果如图 6-34 所示。



```
csu@slave0:~/zookeeper-3.4.9
File Edit View Search Terminal Help
[csu@slave0 zookeeper-3.4.9]$ bin/zkServer.sh status
ZooKeeper JMX enabled by default
Using config: /home/csu/zookeeper-3.4.9/bin/../conf/zoo.cfg
Mode: leader
[csu@slave0 zookeeper-3.4.9]$
```

图 6-34 查看 Slave0 节点上的 ZooKeeper 服务器状态

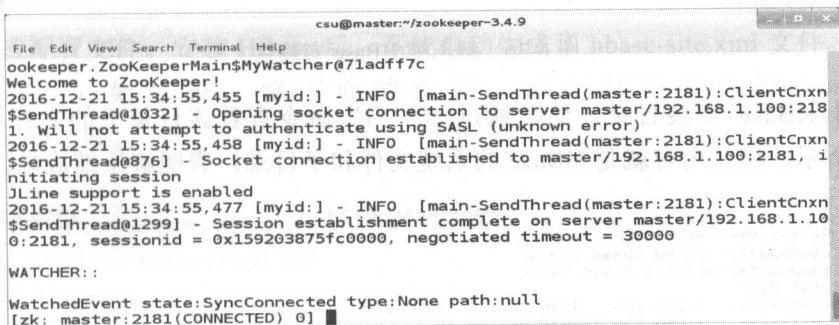
显然，谁是 leader 谁是 follower，这种状态是由系统根据 ZooKeeper 选举机制确定的。

2. ZooKeeper 客户端

ZooKeeper 也提供了一个客户端供用户进行交互式操作。进入 ZooKeeper 客户端的命令是：

```
bin/zkCli.sh -server master:2181
```


注意上述命令中的大小写。执行成功将看到如图 6-35 所示的提示信息，其中 “[zk: master:2181(CONNECTED)0]” 就是客户端命令行提示符。



```

csu@master:~/zookeeper-3.4.9
File Edit View Search Terminal Help
ookeeper.ZooKeeperMain$MyWatcher@71adff7c
Welcome to ZooKeeper!
2016-12-21 15:34:55.455 [myid:] - INFO [main-SendThread(master:2181):ClientCnxn
$SendThread@1032] - Opening socket connection to server master/192.168.1.100:218
1. Will not attempt to authenticate using SASL (unknown error)
2016-12-21 15:34:55.458 [myid:] - INFO [main-SendThread(master:2181):ClientCnxn
$SendThread@876] - Socket connection established to master/192.168.1.100:2181, i
nitiating session
Jline support is enabled
2016-12-21 15:34:55.477 [myid:] - INFO [main-SendThread(master:2181):ClientCnxn
$SendThread@1299] - Session establishment complete on server master/192.168.1.10
0:2181, sessionId = 0x159203875fc0000, negotiated timeout = 30000

WATCHER::

WatchedEvent state:SyncConnected type:None path:null
[zk: master:2181(CONNECTED) 0]

```

图 6-35 ZooKeeper 客户端

读者可以尝试执行几个简单的 ZooKeeper 命令。例如：

- create /zk “MyNode”：创建 znode。
- ls /zk：查看创建的 znode。
- help：帮助命令。
- quit：退出客户端。

6.4 HBase 的安装与应用

HBase 也需要安装在成功部署的 Hadoop 平台上，并且要求 Hadoop 已经正常启动。同时，HBase 需要作为集群部署，因此，我们将在 Master 和 Slave 节点上进行安装。下面的所有操作均使用 csu 用户，请首先进行切换。

6.4.1 解压并安装 HBase

读者可以从 <http://apache.fayea.com/hbase/1.2.4/> 下载 hbase-1.2.4-bin.tar.gz。也可以直接在第 6 章软件资源中，找到 HBase 安装包文件 hbase-1.2.4-bin.tar.gz。

请将该文件复制到 Master 的 “/home/csu/resources” 目录下（为了管理方便，我们建议读者把所有软件资源都放在一个目录下；读者的目录可以不一样），然后将该文件复制到 “/home/csu/” 下面，我们从这里开始安装。

执行 “tar -zxvf hbase-1.2.4-bin.tar.gz” 命令，如图 6-36 所示，系统开始解压缩，滚动显示提示信息。

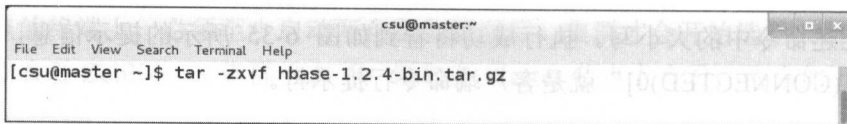


图 6-36 解压缩 HBase 安装包

切换到 HBase 安装目录，然后通过“ls -l”命令查看该目录下的文件和子目录，如图 6-37 所示，这些内容就是 HBase 的系统文件和子目录，表明解压缩成功。

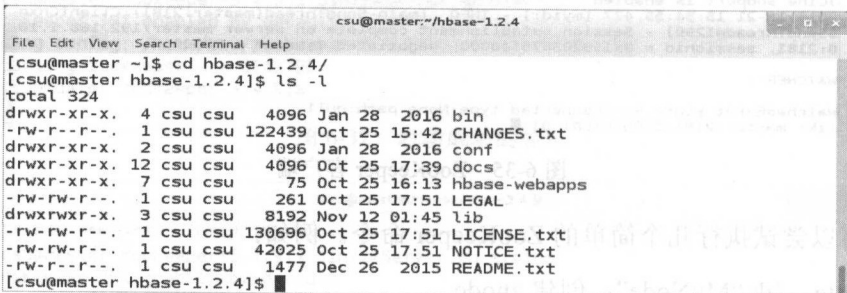


图 6-37 HBase 安装目录

6.4.2 配置 HBase

进入 HBase 安装主目录下的 conf 目录，这是配置文件所在的位置。

1. 修改环境变量 hbase-env.sh

执行“gedit hbase-env.sh”，开始编辑。在打开的文件的靠前部分，可以看到下面的代码。

```
# The java implementation to use. Java 1.7+ required.
# export JAVA_HOME=/usr/java/jdk1.6.0/
```

在上述代码中，修改第二行。去掉“#”号，将“# export JAVA_HOME=/usr/java/jdk1.6.0/”修改为“export JAVA_HOME=/usr/java/jdk1.7.0_71/”，如图 6-38 所示。

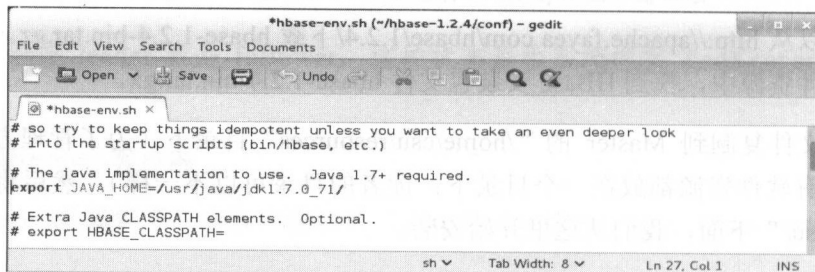


图 6-38 修改 HBase 的环境变量

保存文件，退出 gedit 即可。

2. 修改配置文件 hbase-site.xml

接着修改配置文件。安装 HBase 后，系统自动生成了 hbase-site.xml 文件。执行 gedit hbase-site.xml，开始编辑。将下面的代码，放在打开文件的 <configuration> </configuration> 之间。

```
<property>
    <name>hbase.cluster.distributed</name>
    <value>true</value>
</property>
<property>
    <name>hbase.rootdir</name>
    <value>hdfs://master:9000/hbase</value>
</property>
<property>
    <name>hbase.zookeeper.quorum</name>
    <value>master</value>
</property>
<property>
    <name>hbase.master.info.port</name>
    <value>60010</value>
</property>
```

保存，退出 gedit。

注意事项

必须指出，上述代码中的 60010 是通过 Web 方式查看 HBase 系统状态的端口号。HBase 1.0 以下版本不需要在 hbase-site.xml 文件中添加该端口号，用户就可以访问其 Web 页面。但是 Hbase 1.0 以上版本则需要由用户自己添加，就像我们上面所做的这样。有些用户安装了 Hbase 1.2.4 以后，可以启动并进入 Shell 使用系统，但不能通过 Web 方式查看系统状态，原因就是没有将 hbase-site.xml 文件中添加上述相关配置代码。

3. 设置 regionservers

regionservers 文件类似 Hadoop 的 slaves 文件，其中保存了 regionserver 的列表，启动 HBase 的时候，系统将根据该文件建立 HBase 集群。regionservers 在 HBase 的安装目录下的 conf 子目录中。执行 gedit regionservers，开始编辑，如图 6-39 所示。

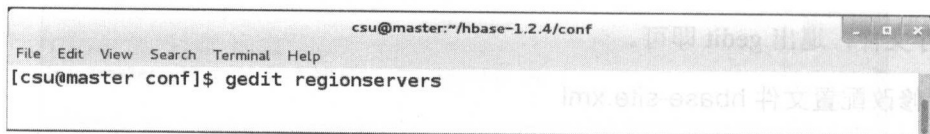


图 6-39 设置 HBase 的 regionservers 文件

在打开的文件中，将已经存在的 localhost 删除，然后添加如下代码：

```
slave0
slave1
```

上述节点列表是本书的情况，读者可根据自己集群实际进行配置。

4. 设置 Linux 环境变量文件

最后修改 .bash_profile 文件。执行 “gedit ~/.bash_profile” 命令，开始编辑，如图 6-40 所示。

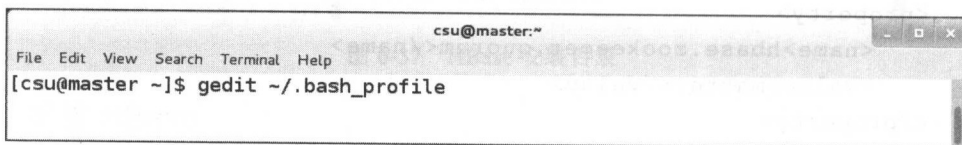


图 6-40 为 HBase 编辑 Linux 用户的配置文件

将下面的代码添加到文件的尾部：

```
#HBase
export HBASE_HOME=/home/csu/hbase-1.2.4/
export PATH=$HBASE_HOME/bin:$PATH
export HADOOP_CLASSPATH=$HBASE_HOME/lib/*
```

编辑完毕，保存退出。然后执行生效命令 “source ~/.bash_profile” 即可，如图 6-41 所示。

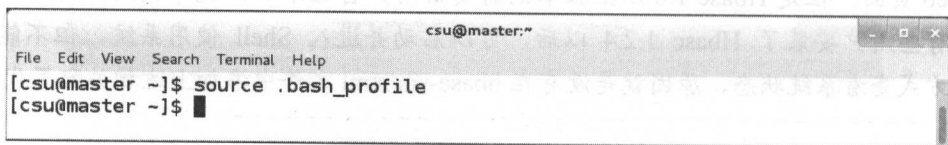


图 6-41 使 .bash_profile 文件生效

5. 将 HBase 安装文件复制到 Slave 节点

本书有两个 Slave 节点 (Slave0 和 Slave1)，因此复制操作需要执行两次。其中一次

的命令是“`scp -r ~/hbase-1.2.4 slave0:~/`”，如图 6-42 所示。

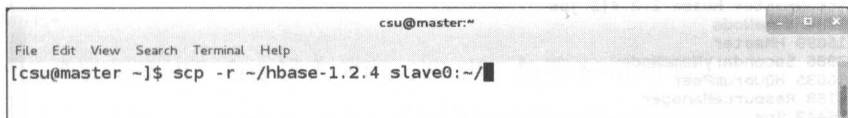


图 6-42 将 HBase 主安装目录复制到 Slave0

回车后开始复制，可以看到终端在滚动显示复制信息，直到复制结束。

至此，HBase 安装与配置进行完毕。

6.4.3 启动并验证 HBase

进入 Master 节点上的 HBase 安装主目录，执行“`bin/start-hbase.sh`”命令，如图 6-43 所示。

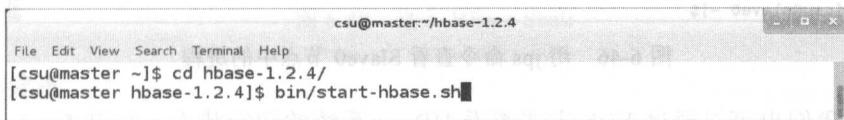


图 6-43 启动 HBase 服务

执行命令后如果看到如图 6-44 所示的打印输出，表明 HBase 已经启动成功。

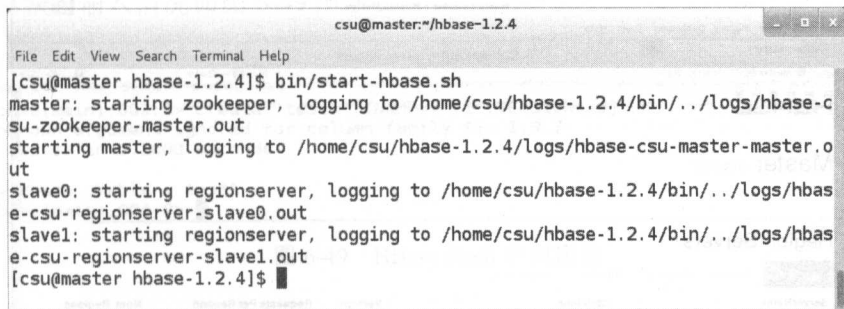


图 6-44 启动成功后的显示信息

可以看到，系统首先启动 HBase 自带的 ZooKeeper，然后启动 HBase 的 HMaster，接着分别启动 Slave 节点上的 regionserver。

读者可以通过 `jps` 命令查看 Master 节点的进程，如图 6-45 所示。其中 HMaster 是 HBase 的主控节点进程，HQuorumPeer 则是 HBase 的 ZooKeeper 进程（即 HBase 内置的 ZooKeeper）。

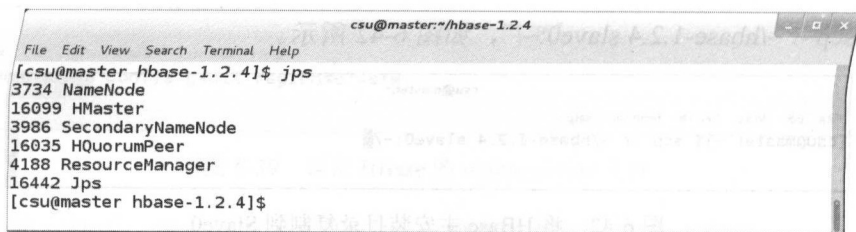


图 6-45 用 jps 命令查看系统中的进程

同时，用户可以在 Slave 节点上用 jps 查看是否存在 HRegionserver 进程，如图 6-46 所示。

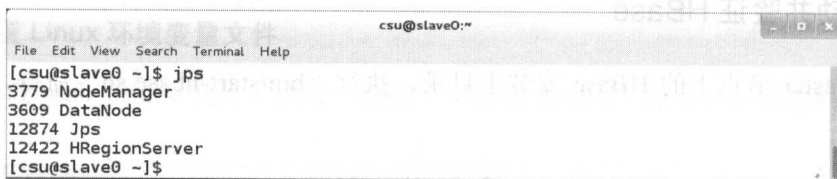


图 6-46 用 jps 命令查看 Slave0 节点中的进程

同样，我们也可以通过 Web 方式查看 HBase 系统的运行状态。打开 Firefox 浏览器，在地址栏输入 <http://master:60010>，会看到如图 6-47 所示的页面，该页面就是 HBase 的管理页面。

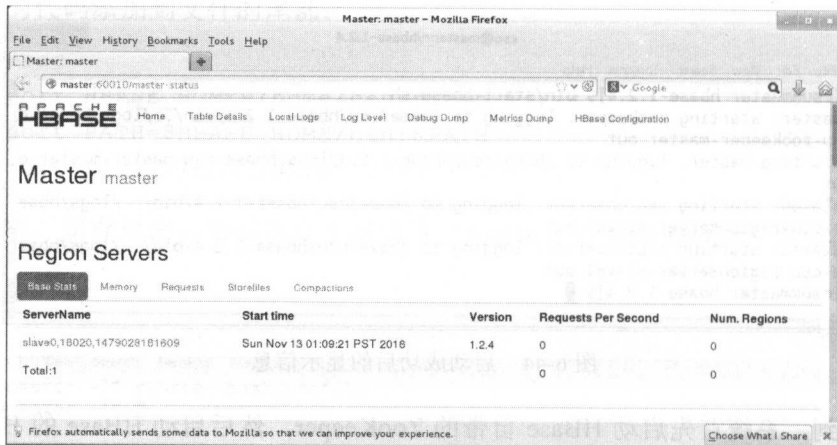


图 6-47 HBase 的 Web 监控页面

6.4.4 HBase 的基本应用

本书将在第 12 章深入介绍 HBase 与其他组件结合的应用，这里我们首先熟悉几个常

见的 HBase Shell 命令。利用 HBase Shell 操作 HBase 是一种基本方法,此外还可以通过程序进行操作。

执行“bin/hbase shell”命令进入 HBase Shell,如图 6-48 所示。

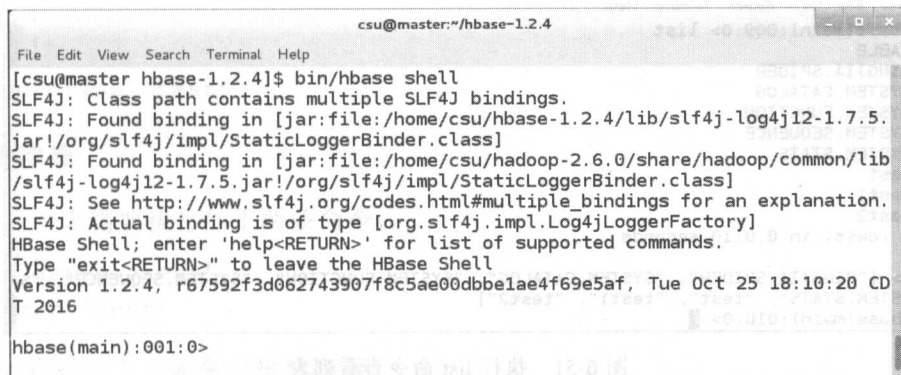


图 6-48 进入 HBase Shell

1. 创建表

命令是:

```
create "test1", {NAME=>'f1', VERSION => 5}
```

上述命令的输入情况如图 6-49 所示。

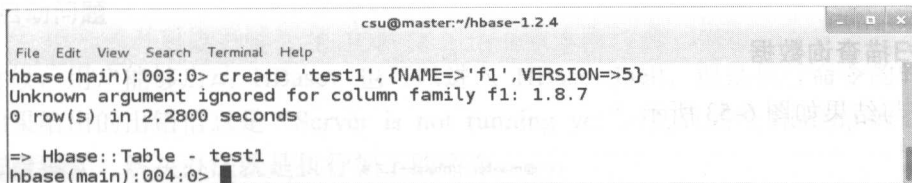


图 6-49 HBase Shell 中创建表

其中 f1 是列族名,由用户自己定义。上述命令是旧版本形式,目前可以简化成如图 6-50 所示的形式。

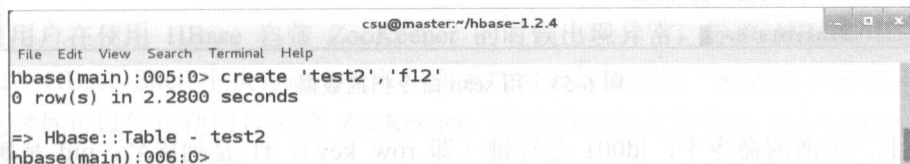
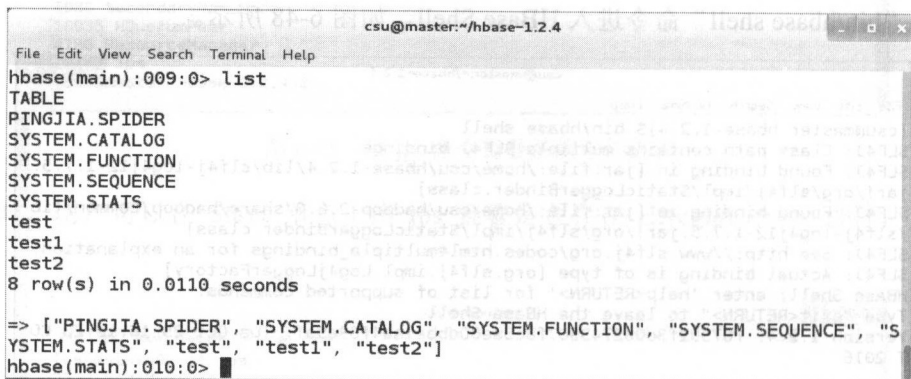


图 6-50 简化的创建表命令

2. 查看列表

命令与执行结果如图 6-51 所示。



```

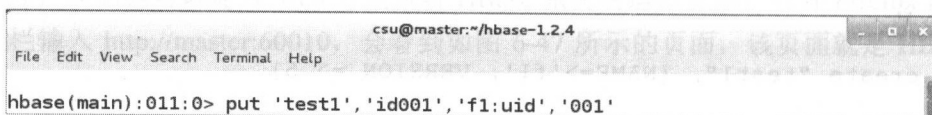
csu@master:~/hbase-1.2.4
File Edit View Search Terminal Help
hbase(main):009:0> list
TABLE
PINGJIA.SPIDER
SYSTEM.CATALOG
SYSTEM.FUNCTION
SYSTEM.SEQUENCE
SYSTEM.STATS
test
test1
test2
8 row(s) in 0.0110 seconds
=> ["PINGJIA.SPIDER", "SYSTEM.CATALOG", "SYSTEM.FUNCTION", "SYSTEM.SEQUENCE", "SYSTEM.STATS", "test", "test1", "test2"]
hbase(main):010:0>

```

图 6-51 执行 list 命令查看列表

3. 插入数据

命令如图 6-52 所示。



```

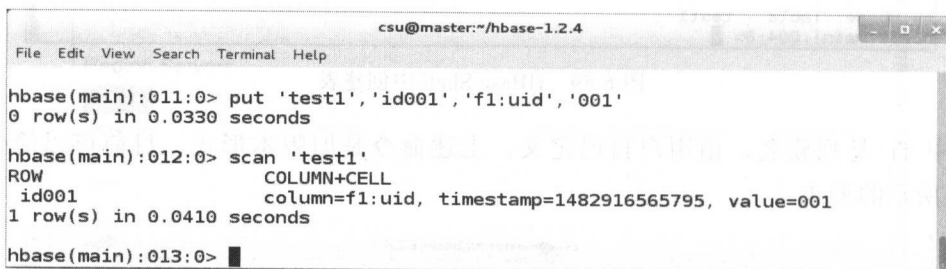
csu@master:~/hbase-1.2.4
File Edit View Search Terminal Help
hbase(main):011:0> put 'test1','id001','f1:uid','001'

```

图 6-52 插入数据命令举例

4. 扫描查询数据

命令与结果如图 6-53 所示。



```

csu@master:~/hbase-1.2.4
File Edit View Search Terminal Help
hbase(main):011:0> put 'test1','id001','f1:uid','001'
0 row(s) in 0.0330 seconds

hbase(main):012:0> scan 'test1'
ROW COLUMN+CELL
id001 column=f1:uid, timestamp=1482916565795, value=001
1 row(s) in 0.0410 seconds
hbase(main):013:0>

```

图 6-53 用 scan 命令扫描数据

实际上，上面的命令中：id001 是行键（即 row key）；f1 是列族名；uid 是属性名，表示用户 ID；而 001 就是属性的值。查询结果中，我们看到了 row key、列族的属性名、

时间戳和属性值。大家不妨对照 HBase 存储模型研究一下，可以看到这里的显示与模型是一致的。

5. 查看表结构

命令是：

```
describe 'test'
```

6. 删除表

删除一个表需要执行如下两条命令：

```
disable 'test'
```

```
drop 'test'
```

用 list 查看一下上述命令执行结果。

7. 退出 HBase Shell

命令是：

```
exit
```

6.4.5 应用 HBase 中常见问题及其解决办法

1. 启动问题

有时候，用户能够启动 HBase，也可以进入 HBase Shell，但是执行命令的时候出现异常，如果给出的出错信息是“Server is not running yet”，其原因是 Hadoop 的 HDFS 集群处于安全模式，解决办法就是执行如下的命令。

```
hadoop dfsadmin -safemode leave
```

重新启动 HBase（当然要先关闭）就可以了。

2. 如何使用独立安装的 ZooKeeper

如果用户在使用 HBase 自带 ZooKeeper 的时候出现异常，导致 HBase 不能正常工作，例如在 HBase Shell 中执行命令失败，而系统给出的提示是“Region Server is not on line”，这时候可以停止使用自带的 ZooKeeper，转而启动独立安装的 ZooKeeper，并重启 Hbase，可快速解决上述问题。

自带的 ZooKeeper 与 HBase 绑定在一起，这种部署模式存在一定的问题。因为当一

个集群中有很多组件都需要 ZooKeeper 的时候，我们面临是启动很多自带的 ZooKeeper，还是采用一个独立安装的 ZooKeeper 的问题，显然后者是合理的方式。

要让 HBase 使用独立安装的 ZooKeeper，需要对 HBase 进行一些配置上的修改。

第一，修改 `conf/hbase-env.s` 文件，添加如下代码：

```
export HBASE_MANAGES_ZK=false
```

上述代码中，如果 “`HBASE_MANAGES_ZK=false`”，表示使用独立安装的 ZooKeeper，等于 `true` 则表示使用自带的 ZooKeeper。

第二，将 ZooKeeper 的配置文件 `zoo.cfg` 复制到 HBase 的 CLASSPATH（此为官方推荐的方式），命令如下：

```
cp /home/csu/zookeeper-2.4.9/conf/zoo.cfg /home/csu/hbase-1.2.4/conf/
```

完成上述配置，重启 HBase 即可。

6.5 本章小结

本章介绍了数据库方面的知识和技术。MySQL 实际上是用于存储结构化数据的关系型数据库，但是它在大数据系统中也有一定作用，大数据其实也需要处理结构化数据，所以 MySQL 集群也是常见的组件。此外，Hive 的元数据存储依赖 MySQL 这样的结构化数据库。正是基于这些事实，我们在本章介绍了 MySQL 的安装与配置。

Hive 是建立在 HDFS 上的数据仓库工具，它为使用 HDFS 与 MapReduce 提供了一种简便的方法，即采用 SQL 语句操作操作 HDFS。

HBase 是典型的 NoSQL，因为它属于列式存储数据库，HBase 也是集群模式的分布式数据库系统。

第 7 章

Sqoop 和 Kafka 的安装与应用

Sqoop 是一个用来完成 Hadoop 和关系型数据库中的数据相互转移的工具，它可以将关系型数据库（MySQL、Oracle、Postgres 等）中的数据导入 Hadoop 的 HDFS 中，也可以将 HDFS 的数据导入关系型数据库中。Kafka 是一个开源的分布式消息订阅系统（消息中间件）。本章主要介绍 Sqoop 和 Kafka 的安装配置及其基本应用方法。

7.1 安装部署 Sqoop

Sqoop 需要安装在成功部署的 Hadoop 平台上，并且要求 Hadoop 已经正常启动。读者可以参见第 6 章中有关验证 Hadoop 是否处于正常运行状态的方法。

准备就绪后，就可以开始安装 Sqoop 了。我们打算将 Sqoop 安装在 Master 节点上，因此以下的操作均在 Master 节点上进行。

7.1.1 下载或复制 Sqoop 安装包

读者可以从 <http://sqoop.apache.org/> 下载到最新版的 Sqoop，也能够在这本书第 7 章软件资源文件夹中，找到 Sqoop 安装压缩包 `sqoop-1.4.6.bin__hadoop-2.0.4-alpha.tar.gz`。

首先将该文件复制到 Master 的“/home/csu”目录下（读者的目录可以不一样）。

注意，安装和运行 Sqoop 都使用 csu 用户，所以需要确保已经切换到 csu 用户，然后进入 /home/csu 目录。

7.1.2 解压并安装 Sqoop

在 /home/csu 目录下，使用命令“tar -zxvf sqoop-1.4.6.bin_hadoop-2.0.4-alpha.tar.gz”，开始解压缩 Sqoop 安装包，如图 7-1 所示。

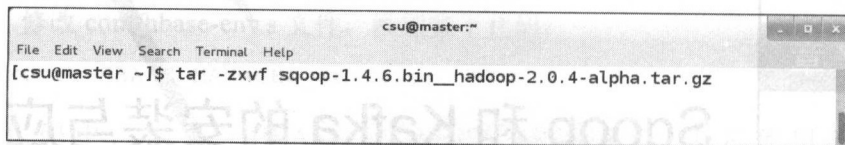


图 7-1 解压缩 Sqoop 安装包

解压缩完毕，系统在“/home/csu/”下创建了 sqoop-1.4.6.bin_hadoop-2.0.4-alpha 目录，该目录即 Sqoop 的安装目录。进入该目录，用“ls -l”命令显示文件和目录列表，如图 7-2 所示，这些文件和目录就是 Sqoop 的系统文件，表明解压缩成功。

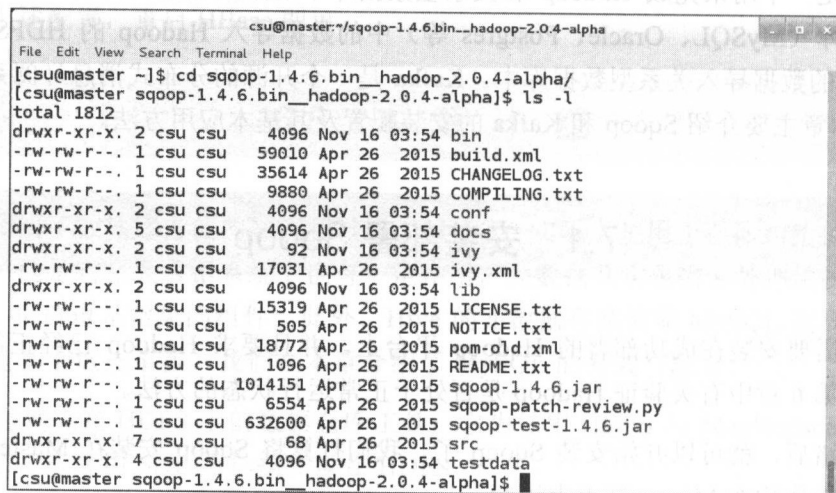


图 7-2 Sqoop 安装目录中的文件及文件夹

7.1.3 配置 Sqoop

1. 配置 MySQL 连接器

Sqoop 经常与 MySQL 结合，帮助从其他数据源向 MySQL 数据库导入数据，或者相反，所以需要配置 MySQL 连接器。读者能够回忆起来，在第 6 章安装 Hive 的时候我们也为其配置过 Java 连接器。实际上，这里也需要进行类似的配置。

所以，我们首先进入“/home/csu/resources/mysql-connector-java-5.1.27/”目录（读者

计算机系统的设置可能不一样), 然后将其中的 `mysql-connector-java-5.1.27-bin.jar` 文件, 复制到 Sqoop 的安装目录的依赖库目录 (即 `lib`) 下, 如图 7-3 所示。

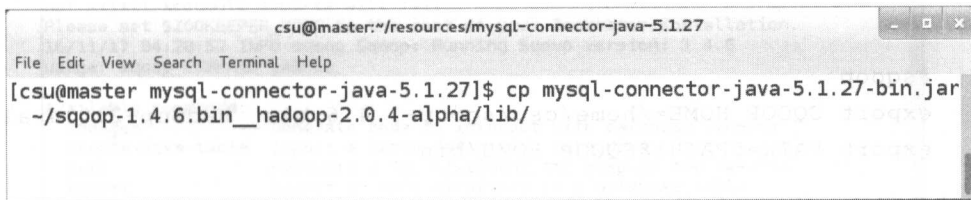


图 7-3 为 Sqoop 安装 Java 连接器

2. 配置 Sqoop 环境变量

在 Sqoop 的主安装目录的 `conf` 子目录下, 系统已经提供了一个环境变量文件模板, 我们首先需要将其名称改为 `sqoop-env.sh`, 然后进行必要的修改。首先进入 Sqoop 的主安装目录的 `conf` 子目录, 然后执行改名操作, 接着用 `gedit` 编辑器打开 `sqoop-env.sh` 文件, 如图 7-4 所示。

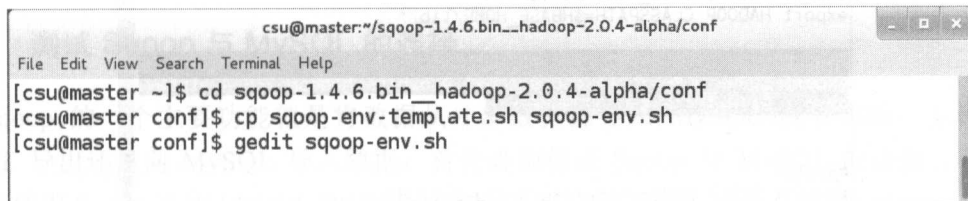


图 7-4 配置 Sqoop 的环境变量

请用下面的代码替换文件中的原有内容:

```

#Set path to where bin/hadoop is available
export HADOOP_COMMON_HOME=/home/csu/hadoop-2.6.0
#Set path to where hadoop-*-core.jar is available
export HADOOP_MAPRED_HOME=/home/csu/hadoop-2.6.0
#set the path to where bin/hbase is available
export HBASE_HOME=/home/csu/hbase-1.2.4
#Set the path to where bin/hive is available
export HIVE_HOME=/home/csu/apache-hive-2.1.0-bin
#Set the path for where zookeeper config dir is
#export ZOOCFGDIR=/usr/local/zk
  
```

编辑完毕, 保存退出。

3. 配置 Linux 环境变量

执行“`gedit /home/csu/.bash_profile`”命令，开始编辑`.bash_profile`文件，输入如下代码。

```
#sqoop
export SQOOP_HOME=/home/csu/sqoop-1.4.6.bin__hadoop-2.0.4-alpha
export PATH=$PATH:$SQOOP_HOME/bin
```

上述代码输入的情况如图 7-5 所示。

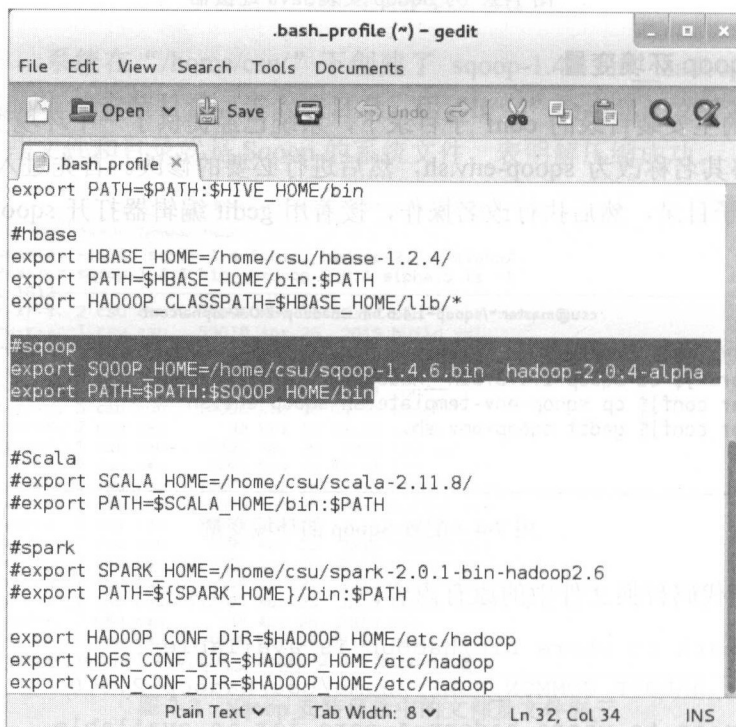


图 7-5 为 Sqoop 配置 Linux 环境变量

编辑完成，保存退出，并使该文件生效（利用 `source /home/csu/.bash_profile` 命令）。

至此，Sqoop 安装配置完毕。

7.1.4 启动并验证 Sqoop

一种简便验证 Sqoop 是否安装成功的办法，就是执行“`bin/sqoop help`”命令，如果看到如图 7-6 所示的显示内容，表示安装成功。

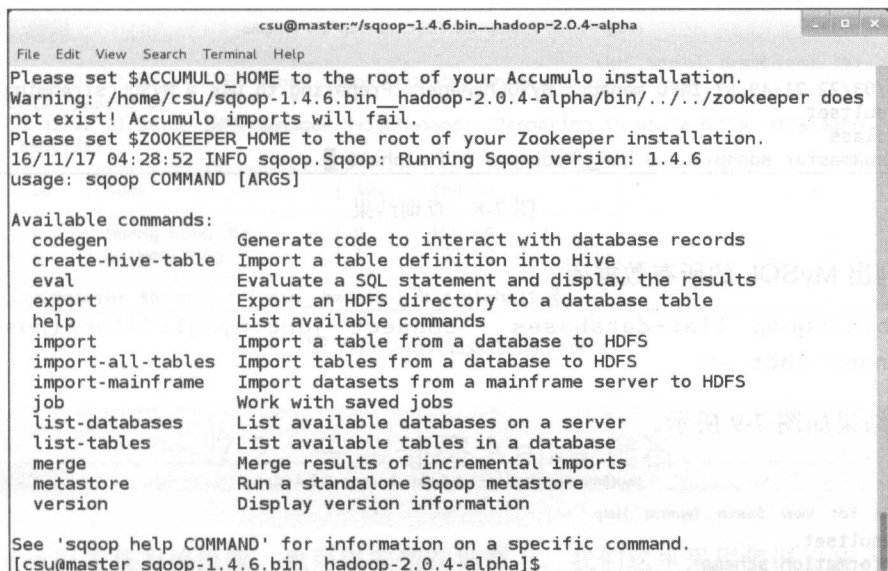


图 7-6 验证 Sqoop 安装是否成功

7.1.5 测试 Sqoop 与 MySQL 的连接

Sqoop 的一个主要功能就是将数据从外部数据源导入 MySQL 或者相反。无论是从 MySQL 导出还是向 MySQL 导入数据，首先必须保证 Sqoop 与 MySQL 的连接。下面我们展示使用 Sqoop 连接 MySQL 数据库的操作。

1. 列出 MySQL 的数据库下的所有表

```
bin/sqoop list-tables --connect jdbc:mysql://localhost:3306/test_db
--username root -P
```

上述命令的输入情形如图 7-7 所示。

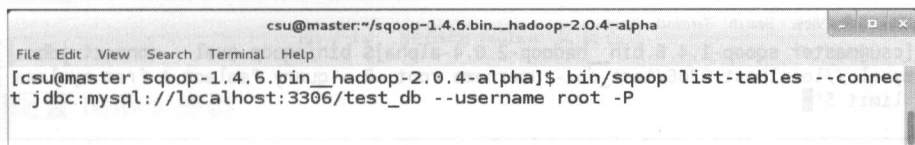


图 7-7 用 Sqoop 列出 MySQL 数据库中的表

上述命令通过 Sqoop 查询 MySQL 下数据库 test_db 的所有表。其中，root 是用户名，后面的 -P 表示需要输入密码，我们这里 root 用户的密码是“qwerty,1289Qwdefi(AUL)”。输入密码回车后，如果执行顺利即可看到查询结果，如图 7-8 所示。

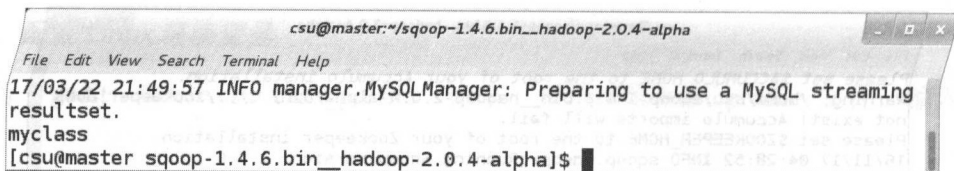


图 7-8 查询结果

2. 列出 MySQL 的所有数据库

```
bin/sqoop list-databases --connect jdbc:mysql://localhost:3306
--username root -P
```

执行结果如图 7-9 所示。

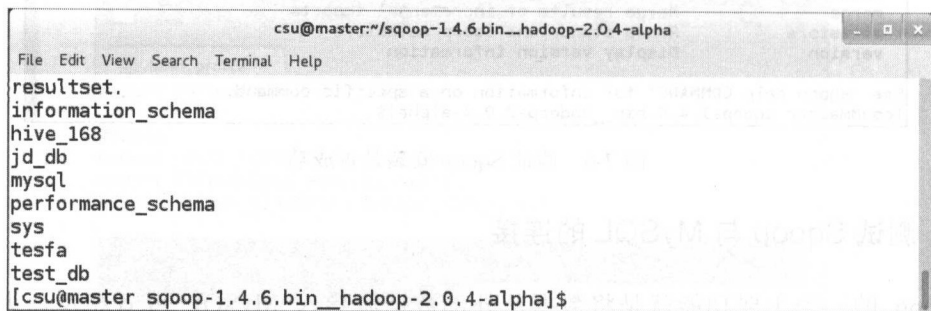


图 7-9 通过 Sqoop 查询 MySQL 的所有数据库

3. 执行查询语句

```
bin/sqoop eval --connect jdbc:mysql://localhost:3306/test_db --
username root -P --query "select * from myclass limit 5"
```

上述命令的输入情形如图 7-10 所示。

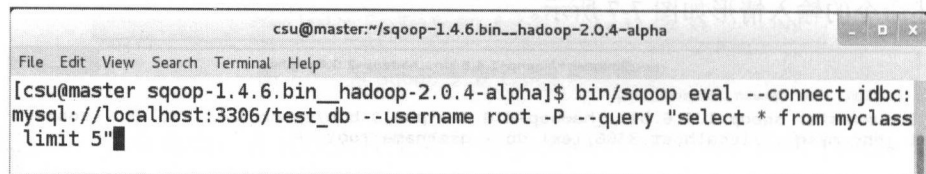
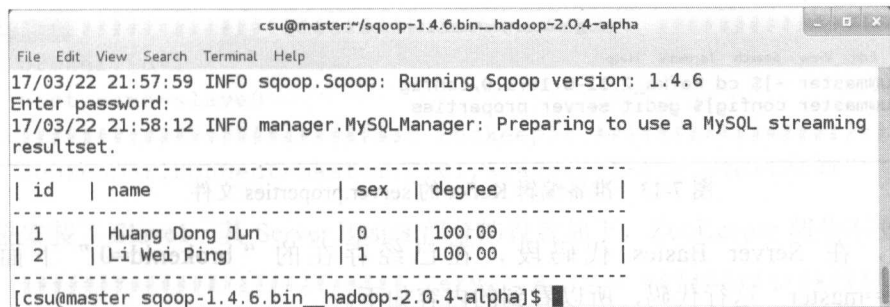


图 7-10 通过 Sqoop 命令查询 MySQL 的 myclass 表

查询结果如图 7-11 所示。

如果能够完成上述操作，说明 Sqoop 能够连接 MySQL，这就为后面传输数据做好了准备。



```
csu@master:~/sqoop-1.4.6.bin__hadoop-2.0.4-alpha
File Edit View Search Terminal Help
17/03/22 21:57:59 INFO sqoop.Sqoop: Running Sqoop version: 1.4.6
Enter password:
17/03/22 21:58:12 INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.
-----
| id | name | sex | degree |
-----
| 1 | Huang Dong Jun | 0 | 100.00 |
| 2 | Li Wei Ping | 1 | 100.00 |
-----
[csu@master sqoop-1.4.6.bin__hadoop-2.0.4-alpha]$
```

图 7-11 myclass 表的记录

7.2 安装部署 Kafka 集群

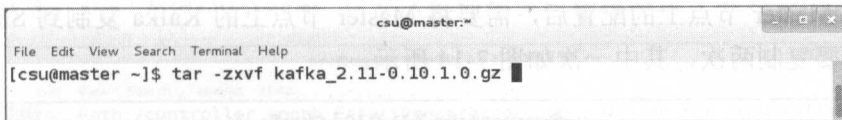
Kafka 可以安装为单机版，也可以安装在集群上。我们这里采用集群安装。

7.2.1 下载或复制 Kafka 安装包

读者可以从 <http://kafka.apache.org/downloads.html> 下载 Kafka 安装包，也可以从本书第 7 章软件资源文件夹中找到 kafka_2.11-0.10.1.0.gz。

7.2.2 解压缩 Kafka 安装包

请 Kafka 安装包文件 kafka_2.11-0.10.1.0.gz 复制到 Master 的“/home/csu/”下，然后进入该目录，并执行解压缩命令“tar -zxvf kafka_2.11-0.10.1.0.gz”，如图 7-12 所示。



```
csu@master:~
File Edit View Search Terminal Help
[csu@master ~]$ tar -zxvf kafka_2.11-0.10.1.0.gz
```

图 7-12 解压缩 Kafka 安装包

7.2.3 配置 Kafka 集群

只需要修改 brokerid 和 zookeeper.connect 项。

1. 在 Master 节点上的配置

在 Master 节点完成如下操作。进入 Kafka 安装主目录下的 config 子目录，然后编辑 server.properties 文件（该文件已经由系统创建）如图 7-13 所示。

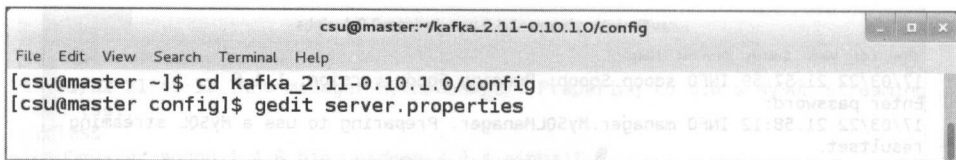


图 7-13 准备编辑 Kafka 的 server.properties 文件

首先，在 Server Basics 代码段，在已经存在的“broker.id=0”下面，增加“host.name=master”这行代码，所以看到的内容如下。

```
broker.id=0
host.name=master
```

保存修改。

显然，上述配置选择了 master 作为 broker，其 id 采用了默认的 0，所以保持不变，仅增加 master 的主机名，这也是为了名称解析的需要。

接着，我们还是在该文件的 ZooKeeper 代码段，将已有的“zookeeper.connect=localhost:2181”替换成如下的代码（根据实际集群情况配置）。

```
zookeeper.connect=master:2181,slave0:2181,slave1:2181
```

保存修改，退出 gedit。

可以看出，ZooKeeper 作为协调器，它连接的节点包括了集群的所有计算机。这里的配置显然需要根据用户自己的集群情况配置，如果仅安装了一个 Slave 节点（本书是 Slave0），就只需要填写 slave0:2181 即可。

完成了 Master 节点上的配置后，需要将 Master 节点上的 Kafka 复制到 Slave 节点。我们这里需要复制两次，其中一次如图 7-14 所示。

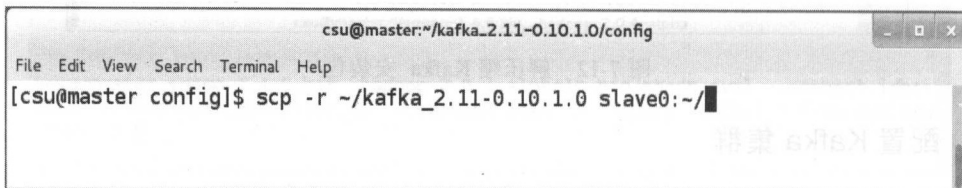


图 7-14 将 Master 节点上的 Kafka 安装目录复制到 Slave0 节点

2. 在 Slave 节点上的配置

接着还需要在 Slave 节点上进行必要的配置。对于这里的 Slave0，将其 server.properties 配置文件中的 broker.id 设置为 1，host.name 设置为 Slave0，ZooKeeper 部分保持不变：


```
##### Server Basics #####
broker.id=1
host.name=slave0
##### Zookeeper #####
zookeeper.connect=master:2181,slave0:2181,slave1:2181
```

如果安装了 Slavel, 其 Server Basics 部分的设置如下, ZooKeeper 部分不变。

```
##### Server Basics #####
broker.id=2
host.name=slave1
##### Zookeeper #####
zookeeper.connect=master:2181,slave0:2181,slave1:2181
```

至此, Kafka 集群的配置就完成了。

7.2.4 Kafka 的初步应用

1. 启动 ZooKeeper 服务

要使用 Kafka, 首先需要启动 ZooKeeper 服务, 进入 Kafka 安装主目录, 执行如下命令:

```
bin/zookeeper-server-start.sh config/zookeeper.properties
```

执行上述命令后, 终端显示一些信息, 如图 7-15 所示。

显然, 我们这里启动了 Kafka 自带的 ZooKeeper 服务器。注意, 执行上述命令后, 终端显示一些信息后, 就出现停顿, 这并非表示 ZooKeeper 服务启动失败, 而是系统正处于后台运行状态, 用户无须特别操作, 只要保持终端窗口状态即可 (不要关闭)。

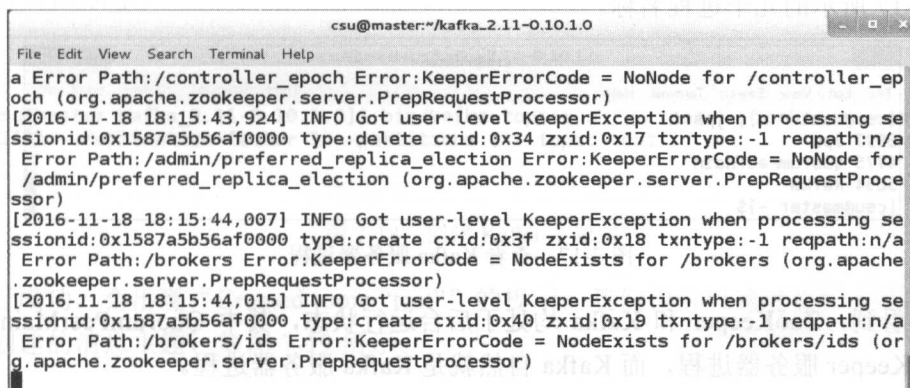
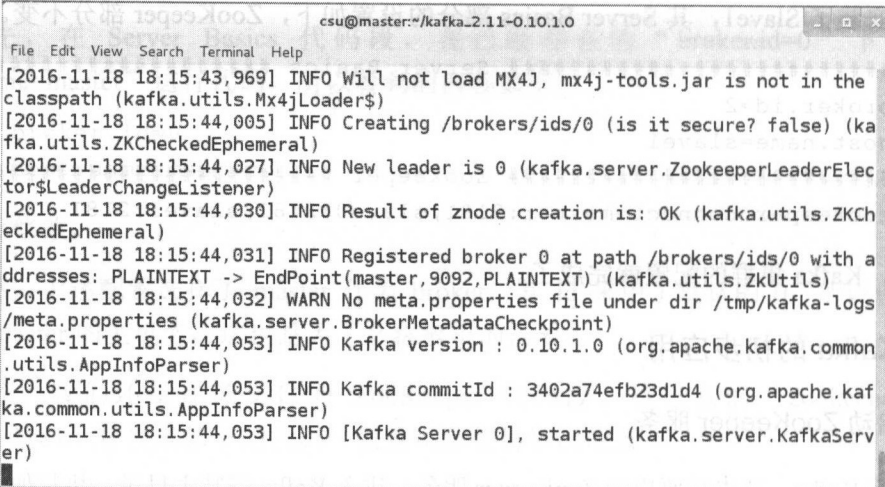


图 7-15 启动 ZooKeeper 服务

2. 启动 Kafka 服务

接着, 我们分别在 Master 节点和 Slave 节点启动 Kafka 服务集群。首先启动 master 节点上的 Kafka 服务。新开辟一个终端, 然后进入 Kafka 主目录, 执行“bin/kafka-server-start.sh config/server.properties”命令, 系统显示如图 7-16 所示。

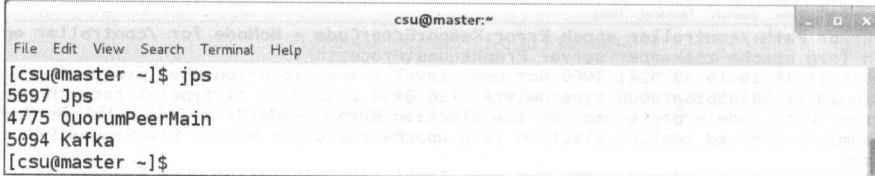


```
csu@master:~/kafka_2.11-0.10.1.0
File Edit View Search Terminal Help
[2016-11-18 18:15:43,969] INFO Will not load MX4J, mx4j-tools.jar is not in the
classpath (kafka.utils.Mx4jLoader$)
[2016-11-18 18:15:44,005] INFO Creating /brokers/ids/0 (is it secure? false) (ka
fka.utils.ZKCheckedEphemeral)
[2016-11-18 18:15:44,027] INFO New leader is 0 (kafka.server.ZookeeperLeaderElec
tor$LeaderChangeListener)
[2016-11-18 18:15:44,030] INFO Result of znode creation is: OK (kafka.utils.ZKCh
eckedEphemeral)
[2016-11-18 18:15:44,031] INFO Registered broker 0 at path /brokers/ids/0 with a
ddresses: PLAINTEXT -> EndPoint(master,9092,PLAINTEXT) (kafka.utils.ZkUtils)
[2016-11-18 18:15:44,032] WARN No meta.properties file under dir /tmp/kafka-logs
/meta.properties (kafka.server.BrokerMetadataCheckpoint)
[2016-11-18 18:15:44,053] INFO Kafka version : 0.10.1.0 (org.apache.kafka.common
.utils.AppInfoParser)
[2016-11-18 18:15:44,053] INFO Kafka commitId : 3402a74efb23d1d4 (org.apache.kaf
ka.common.utils.AppInfoParser)
[2016-11-18 18:15:44,053] INFO [Kafka Server 0], started (kafka.server.KafkaServ
er)
```

图 7-16 启动 Kafka 服务器

同样, 由于 Kafka 是作为守护进程加载的, 执行上述命令后终端显示出现停顿状态, 这表示系统已经处于后台运行状态, 所以, 也不需要关闭该终端, 只要保持当前状态即可。

实际上, 用户可以另外开一个终端, 执行 jps 命令, 查看当前系统进程列表, 可以看到如图 7-17 所示的几个进程名称。



```
csu@master:~
File Edit View Search Terminal Help
[csu@master ~]$ jps
5697 Jps
4775 QuorumPeerMain
5094 Kafka
[csu@master ~]$
```

图 7-17 查看 Kafka 服务器进程

可以看到, ZooKeeper 和 Kafka 均处于后台运行状态, 其中 QuorumPeerMain 就是自带的 ZooKeeper 服务器进程, 而 Kafka 自然就是 Kafka 服务器进程。

同样我们可以在 Slave 节点上启动 Kafka 服务。

注意事项

有一些用户可能在启动 Kafka 自带的 ZooKeeper 服务的时候失败，或者即使启动了，也不能正常工作，例如无法支持主题创建（下面马上就会介绍）。

这时候，我们可以先关闭刚才启动的 ZooKeeper 服务（可以简单地通过 `kill -9 PID` 命令终止 QuorumPeerMain 服务，注意其中 PID 是 QuorumPeerMain 的进程 ID 号，例如上面示例中的 4775）。然后通过 HBase 来启动其自带的 ZooKeeper 服务，或者启动独立安装的 ZooKeeper。只要启动成功，也同样可以为 Kafka 的工作提供支持。

上述实践说明，在 Hadoop 集群应用中，我们只要能够启动任何一个组件自带的 ZooKeeper 服务，或者启动独立安装的 ZooKeeper，就可以为其他任何需要 ZooKeeper 服务的组件提供支持了，并不需要分别启动自带的 ZooKeeper。

3. 创建主题

要使用 Kafka，一定需要创建消息主题。主题（Topic）是消息中间件的基本概念，相当于文件系统的目录，其实就是用于保存消息内容的计算实体，通过主题名称加以标识，就如同目录通过目录名标识一样。

我们在 Master 节点上创建一个名为 `test` 的主题。注意，为了创建主题，请在 Master 上另外开辟一个终端，进入 Kafka 主安装目录，并执行如下命令：

```
bin/kafka-topics.sh --create --zookeeper master:2181--replication-factor  
1 --partitions 1 --topic test
```

如图 7-18 所示。

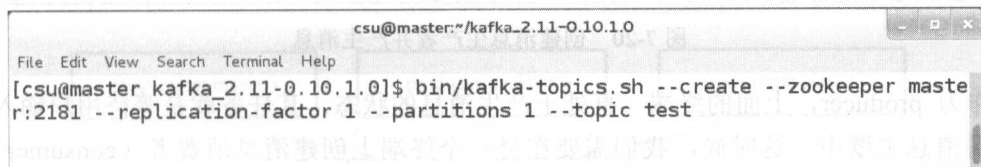


图 7-18 创建 Kafka 主题

执行成功，会出现“Created topic “test””的提示。我们也可以通过执行下面的命令查看已经创建的消息主题，如图 7-19 所示。

```
bin/kafka-topics.sh --list --zookeeper master:2181
```

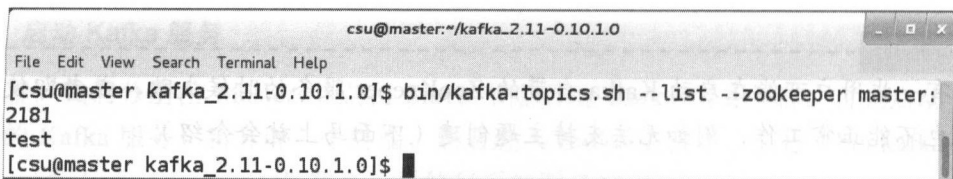


图 7-19 查看创建的主题

4. 发送消息

消息中间件是一个用于接收消息并转发消息的服务，所以，为了检验 Kafka 是否能够正常工作，我们需要创建一个消息生产者（producer），利用该它产生消息。请重新开辟一个终端，然后执行如下的命令：

```
bin/kafka-console-producer.sh --broker-list master:9092 --topic test
```

回车后，系统等待用户输入信息，例如可以输入 “This is a message from the terminal atmaster” 等，如图 7-20 所示。

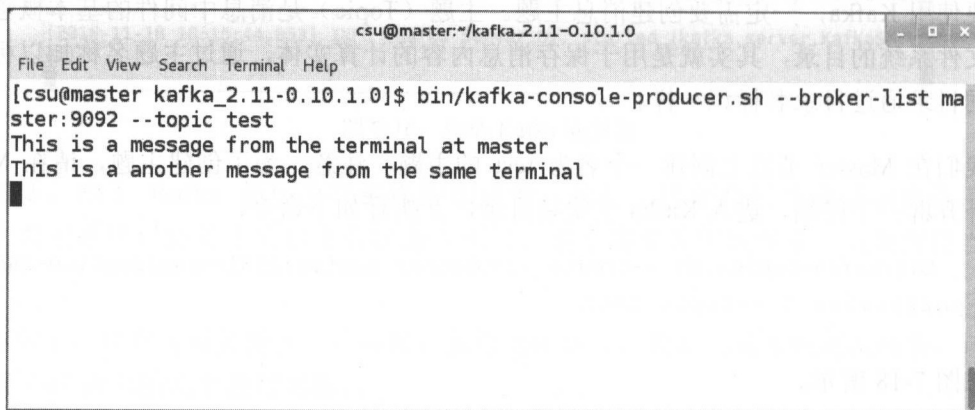


图 7-20 创建消息生产者并产生消息

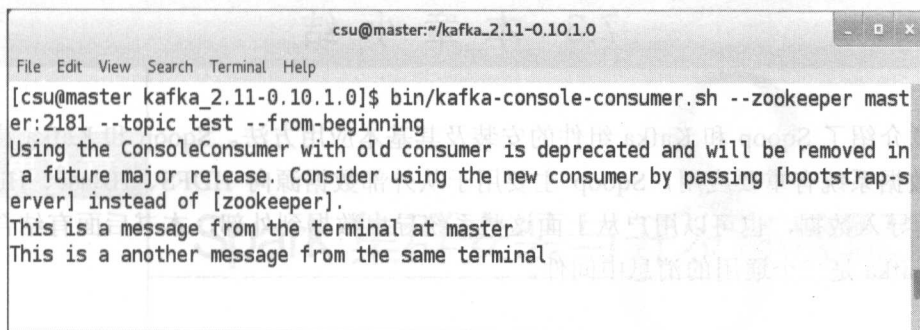
作为 producer，上面的终端一直处于产生消息的状态，其任务就是等待用户输入，并保存到消息主题中。这时候，我们需要在另一个终端上创建消息消费者（consumer），才能接收这些消息。

5. 消费消息

要创建消费者并接收消息，需要在一个新终端上执行如下的命令：

```
bin/kafka-console-consumer.sh --zookeeper master:2181 --topic test --from-beginning
```

回车后，即可接收到从生产者发送到 test 主题中的消息，如图 7-21 所示。



```

csu@master:~/kafka_2.11-0.10.1.0
File Edit View Search Terminal Help
[csu@master kafka_2.11-0.10.1.0]$ bin/kafka-console-consumer.sh --zookeeper mast
er:2181 --topic test --from-beginning
Using the ConsoleConsumer with old consumer is deprecated and will be removed in
a future major release. Consider using the new consumer by passing [bootstrap-s
erver] instead of [zookeeper].
This is a message from the terminal at master
This is a another message from the same terminal
  
```

图 7-21 创建消费者并接收消息

要查看主题中的信息，执行如下命令：

```
bin/kafka-topics.sh --describe --zookeeper master:2181 --topic test
```

可以看到，消息生产者和消费者通过 Kafka 中间件联系起来了，生产者是产生消息的一方，而消费者只需要从主题中接收消息。这样一种应用模式，在很多数据处理系统中都可以发挥积极作用。例如，在一些实时大数据应用中，Kafka 可以保存从数据源产生的数据，接收者（消费者）则可以采用自己的数据率接收数据，因此，Kafka 起到了一个缓冲作用。

由于 Kafka 是一个分布式的消息分发系统，所以，也可以在集群中的任何节点接收消息，例如在 Slave0 节点上，通过在终端执行“bin/kafka-console-consumer.sh --zookeeper master:2181 --topic test --from-beginning”命令，也能够接收到从 Master 的终端发来的消息。同理，我们还可以在 Slave 节点上创建消息生产者，向 Kafka 服务器发送消息，使得集群的任何节点都可以接收消息。图 7-22 给出了一个简单的消息分发架构。

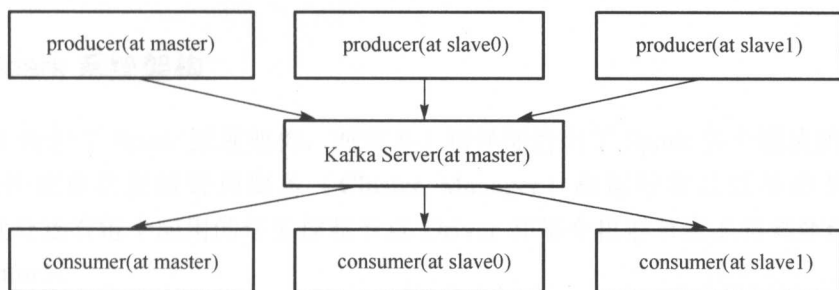


图 7-22 一个消息分发架构示例

7.3 本章小结

本章介绍了 Sqoop 和 Kafka 组件的安装及其基本应用方法。Sqoop 和 Kafka 这两个组件在大数据系统有重要应用，Sqoop 主要用于从外部数据源向 HDFS、HBase、Hive 甚至 MySQL 导入数据，也可以用户从上面这些系统导出数据到外部，本书后面有结合项目的应用。Kafka 是一个通用的消息中间件。



随着 Yarn 的出现, MapReduce 使用者不再需要担心任务的并行性和容错问题, 只需要使用一些基本的操作就能并行地读写数据。但是, 由于 MapReduce 框架并没有很好地使用分布式内存, 每个 MapReduce 任务需要读写磁盘, 这使得 MapReduce 对于某些需要重用中间结果的应用很低效。在很多迭代式的机器学习和数据挖掘算法中, 使用中间结果是非常常见的。如果使用 MapReduce 框架来处理这种类型的应用, 那么数据副本、磁盘 I/O 和数据序列化将会花费大量时间。因为内存的读写速度远远高于磁盘, 在理想状况下, 如果所有的工作数据都能放入内存, 那么大部分的任务就能在很短的时间内完成。

为了避免 MapReduce 框架中多次读写磁盘的消耗, 更充分地利用内存, UC Berkeley AMP Lab 提出了一种新的开源的类 Hadoop MapReduce 的内存编程模型, 这就是 Spark。

8.1 深入理解 Spark

8.1.1 Spark 系统架构

图 8-1 给出了 Spark 原理架构, 而表 8-1 则详细给出了 Spark 各个模块的说明。Spark 的核心组件是集群资源管理服务 (Cluster Manager) 和运行作业任务的节点 (Worker Node), 此外还有每个应用的任务控制节点 Driver 和每个机器节点上有具体任务的执行进程 (Executor)。

与 MR 计算框架相比, Executor 有两个优点: 一个是采用多线程来执行具体的任务, 而不是像 MR 那样采用进程模型, 减少了任务的启动开销; 二是 Executor 上会有一个

Block Manager 存储模块，类似于 KV 系统（内存和磁盘共同作为存储设备），当需要多轮迭代时，可以将中间过程的数据先放到这个存储系统上，下次需要时直接读该存储上数据，而不需要读写到 HDFS 等相关的文件系统里；或者在交互式查询场景下，事先将数据表缓存到该存储系统上，从而提高读写 IO 性能。此外，Spark 在做 Shuffle 时，在 Groupby、Join 等场景下去掉了不必要的 Sort 操作；同时，相比于 MapReduce 只有 Map 和 Reduce 两种模式，Spark 还提供了更加丰富全面的运算操作，如 Filter、Groupby、Join 等。

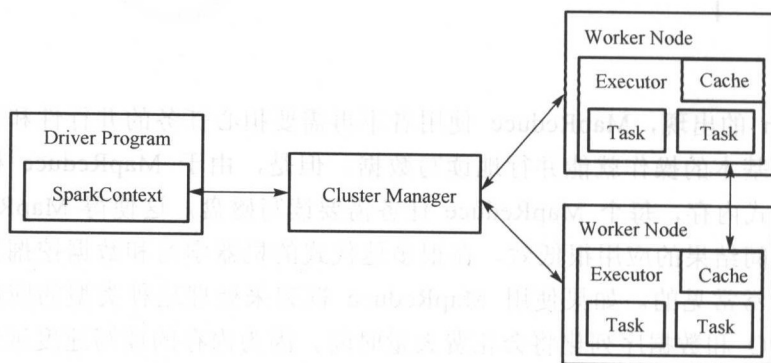


图 8-1 Spark 原理架构

表 8-1 Spark 中模块的说明

模 块	说 明
Driver Program	Spark 应用程序运行时包含一个 Driver 进程，也是应用程序的主进程，负责应用程序的解析、生成 Stage 并调度 Task 到 Executor 上
Cluster Manager	集群管理器。Spark 支持多种集群管理器，包括 Spark 自带的 Standalone 集群管理器、Mesos 或 Yarn
Executor	即真正执行应用程序的地方，一个集群一般包含多个 Executor，每个 Executor 接收 Driver 的命令 Launch Task，一个 Executor 可以执行一到多个 Task
Master Node	集群的主节点，负责接收客户端提交的作业，管理 Worker，并命令 Worker 启动 Driver 和 Executor
Work Node	负责运行集群中的 Spark 应用程序。负责管理本节点的资源，定期向 Master 汇报心跳，接收 Master 的命令
SparkContext	面向用户的 Spark 程序入口，是 Spark 应用程序的总控，负责运用的分布执行计划和 Task 的调度。是用户基于业务逻辑定义的类，里面包含 DAG（无回路有向图）。SparkContext 会基于用户的业务逻辑，划分 Stage，并生成 Task
Task	承载业务逻辑的运算单元，是 Spark 平台中可执行的最小工作单元。一个应用根据执行计划以及计算量分为多个 Task
Cache	分布式缓存。每个 Task 可将结果放置于 Cache，供多个后续 Task 读取

Spark 应用的执行过程如下。

(1) 构建 Spark Application 的运行环境，启动 SparkContext。

(2) SparkContext 向资源管理器（可以是 Standalone、Mesos、Yarn）申请运行 Executor 资源，并启动 StandaloneExecutorbackend。

(3) Executor 向 SparkContext 申请 Task。

(4) SparkContext 将应用程序分发给 Executor。

(5) SparkContext 构建 DAG 图，将 DAG 图分解成 Stage，将 Taskset 发送给 Task Scheduler，最后由 Task Scheduler 将 Task 发送给 Executor 运行。

(6) Task 在 Executor 上运行，运行完释放所有资源。

8.1.2 关键概念

1. RDD

Spark 的核心概念是弹性分布式数据集（Resilient Distributed Datasets, RDD）。RDD 是一个只读且不可变的分布式对象集合，创建、转化及调用 RDD 操作这一系列过程贯穿于 Spark 大数据处理的始终。在底层，Spark 将 RDD 中的数据分发到集群上，进行并行化操作。为了保证高效的容错性，RDD 维护一系列操作的血统信息，即记录下 RDD 间转化的操作过程。当出现故障时可以利用血统信息并行恢复丢失的数据，而不需要重新进行整个计算。

作用在 RDD 上的操作分为两类：Transformation 和 Action。Transformation 操作会由一个 RDD 生成一个新的 RDD，数据集中的内容会发生改变。Action 操作会对 RDD 计算出一个结果并返回到驱动器程序中。虽然二者都是对 RDD 进行操作，但是只有当对 RDD 进行 Action 操作时，该 RDD 及其父 RDD 上的所有操作才会提交到集群中进行真正的计算，这称为惰性计算（Lazy Computing）。

2. DAG

Spark 使用有向无环图（DAG 图）进行任务调度，图 8-2 是 Spark 调度运行机制示意。

Spark 应用程序提交后会经历一系列的转换，最后成为任务在集群的各个节点上执行。当某个 RDD 有 Action 操作时，该 Action 会作为一项工作提交。在提交阶段，DAG 调度模块会检查 RDD 的血统图，计算各 RDD 间的依赖关系，并根据这些依赖关系，将整个工作分割成为若干个不同的阶段（Stages），在各个阶段被提交之后，由 Task 调度模块根据阶段来计算所需要进行的任务，并将这些任务分发到对应的计算节点。

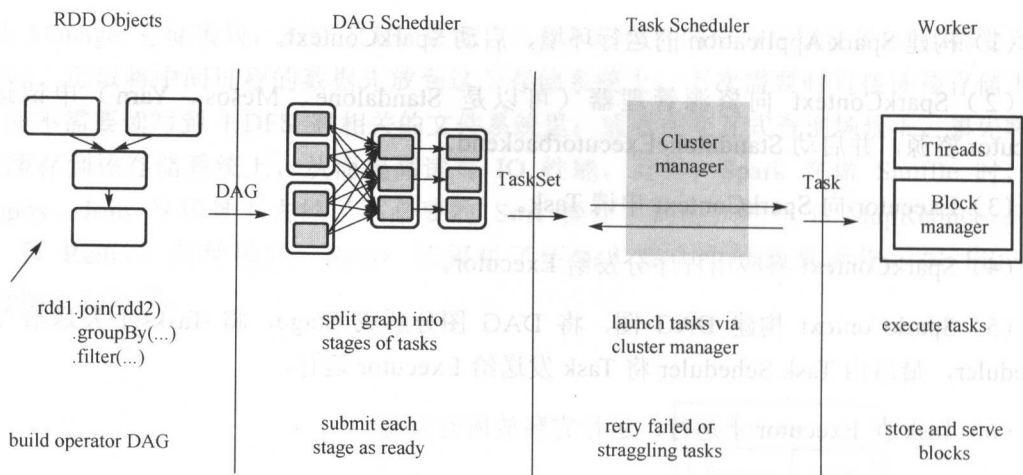


图 8-2 Spark 的调度运行机制

3. Spark SQL

Spark SQL 是 Spark 的一个组件，用于结构化数据的计算。Spark SQL，作为 Apache Spark 大数据框架的一部分，主要用于结构化数据处理和对 Spark 数据执行类 SQL 的查询。通过 Spark SQL，可以针对不同格式的数据执行 ETL 操作，如 JSON、Parquet（一种面向分析型业务的列式存储格式）和关系型数据库，然后完成特定的查询操作。

Spark SQL 提供了十分友好的 SQL 接口，可以与来自多种不同数据源的数据进行交互。除了文本文件之外，也可以从其他数据源中加载数据，如 JSON 数据文件、Hive 表，甚至可以通过 JDBC 数据源加载关系型数据库表中的数据。由于采用的语法是熟知的 SQL 语言，这对于非技术类的项目成员，如数据分析师以及数据库管理员来说，也非常实用。

使用 Spark SQL 时，最主要的两个组件就是 `DataFrame` 和 `SQLContext`。

4. DataFrame

`DataFrame` 是一个分布式的、按照命名列的形式组织的数据集合。`DataFrame` 基于 R 语言中的 `Data Frame` 概念，与关系型数据库中的数据库表类似。之前版本的 Spark SQL API 中的 `SchemaRDD` 已经更名为 `DataFrame`。通过调用将 `DataFrame` 的内容作为行 RDD（RDD of Rows）返回的 `RDD` 方法，可以将 `DataFrame` 转换成 `RDD`。

可以通过已有的 `RDD`、结构化数据文件、JSON 数据集、Hive 表和外部数据库等数据源创建 `DataFrame`。

Spark SQL 和 DataFrame API 已经在 Scala、Java、Python 和 R 程序设计语言中实现。

5. SQLContext

Spark SQL 提供 SQLContext 封装 Spark 中的所有关系型功能，可以用前面提到的 SparkContext 创建 SQLContext。下述代码片段展示了如何创建一个 SQLContext 对象。

```
val sqlContext = new org.apache.spark.sql.SQLContext(sc)
```

此外，Spark SQL 中的 HiveContext 可以提供 SQLContext 所提供功能的超集，可以在用 HiveQL 解析器编写查询语句，以及从 Hive 表中读取数据时使用，在 Spark 程序中使用 HiveContext 无须既有的 Hive 环境。

6. JDBC 数据源

Spark SQL 库的其他功能还包括数据源，如 JDBC 数据源。

JDBC 数据源可用于通过 JDBC API 读取关系型数据库中的数据。相比于使用 JdbcRDD，应该将 JDBC 数据源的方式作为首选，因为 JDBC 数据源能够将结果作为 DataFrame 对象返回，直接用 Spark SQL 处理或与其他数据源连接。

7. Spark 和 HDFS 的配合关系

通常，Spark 中计算的数据可以来自多个数据源，如 local File、HDFS 等。最常用的是 HDFS，可以一次读取大规模的数据进行并行计算。在计算完成后，也可以将数据存储到 HDFS。

分解来看，Spark 分成控制端（Driver）和执行端（Executor），控制端负责任务调度，执行端负责任务执行。读取文件的过程如图 8-3 所示。

（1）读取文件的详细步骤：

- SparkScheduler 与 HDFS 交互获取 File A 的文件信息。
- HDFS 返回该文件具体的 Block 信息。
- SparkScheduler 根据具体的 Block 数据量，决定一个并行度，创建多个 Task 去读取这些文件 Block。
- Executor 端执行 Task 并读取具体的 Block，作为 RDD（弹性分布数据集）的一部分。

（2）HDFS 文件写入的详细步骤：

- SparkScheduler 创建要写入文件的目录。

- 根据 RDD 分区分块情况, 计算出写数据的 Task 数, 并下发这些任务到 Executor。
- Executor 执行这些 Task, 将具体 RDD 的数据写入到第一步创建的目录下。

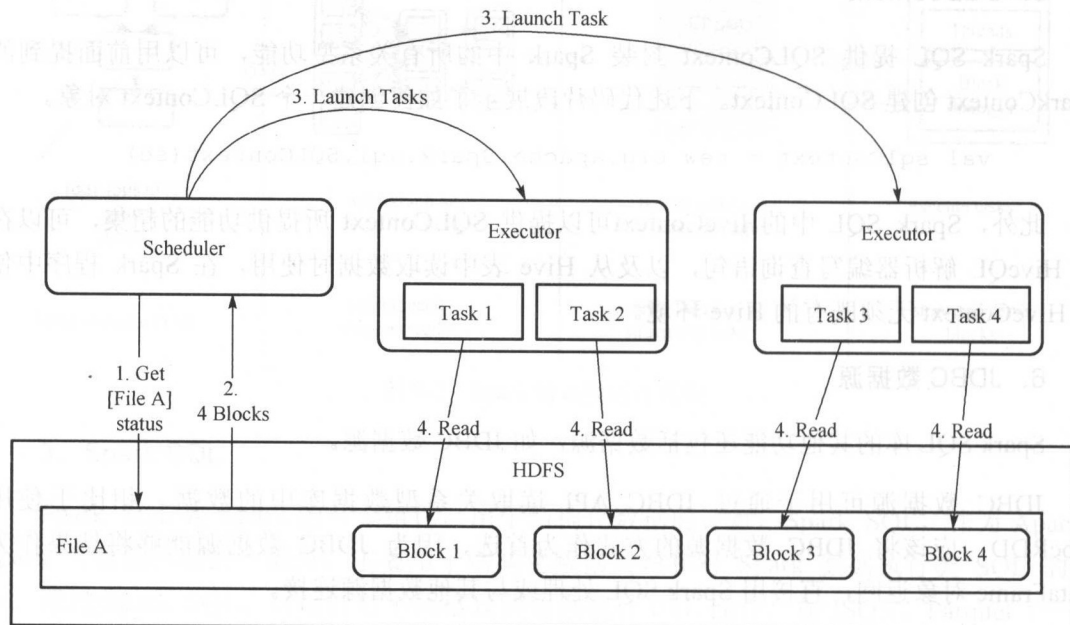


图 8-3 Spark 读文件过程

8.2 安装与配置 Scala

在安装 Spark 之前, 我们首先要安装一种编程语言环境, 这就是 Scala。我们知道, Spark 是用 Scala 语言实现的, 而且主要支持 Scala 语言进行应用开发 (当然也支持其他语言, 如 Java)。因此, 为了帮助读者熟悉 Spark 平台上的 Scala 程序设计, 我们搭建的 Spark 开发环境也需要安装 Scala。

Scala 是一种纯面向对象的语言, 每一个值都是对象。对象的数据类型及行为由类和特征描述。类抽象机制的扩展有两种途径, 一种途径是子类继承, 另一种途径是灵活的混入 (Mixin) 机制, 这两种途径能避免多重继承的种种问题。

Scala 也是一种函数式语言, 提供轻量级的语法用以定义匿名函数, 支持高阶函数, 允许嵌套多层函数。Scala 的 Case Class 及其内置的模式匹配相当于函数式编程语言中常用的代数类型 (Algebraic Type)。

在函数表达上, Scala 有天然的优势, 因此, 表达复杂的机器学习算法的能力比其他语言更强且简单易懂。Spark 提供各种操作函数来建立起 RDD 的 DAG 计算模型。Spark 把每一个操作都看成构建一个 RDD 来对待, 而 RDD 则表示的是分布在多台机器上的数据集集合, 并且可以带上各种操作函数。

Scala 运行于 Java 平台 (Java 虚拟机), 并兼容现有的 Java 程序, 它也能运行于 Java ME、CLDC (Micro Edition Connected Limited Device Configuration) 上。

8.2.1 下载 Scala 安装包

由于本书后面安装的 Spark 是 2.0.1 版本, 而该版本的 Spark 建议使用 2.11.0 以上的 Scala, 因此, 请读者首先从网络上下载合适的 Scala 版本, 如 scala-2.11.8.tgz。读者也可以在本书第 8 章软件资源中找到 Scala 安装包 scala-2.11.8.tgz。

请将 scala-2.11.8.tgz 文件复制到 Master 节点的 “/home/csu/” 目录下。

8.2.2 安装 Scala

进入 “/home/csu/” 目录, 执行 “tar -zxvf scala-2.11.8.tgz”, 如图 8-4 所示, 解压缩该安装包, 系统生成 Scala 的主安装目录 scala-2.11.8。

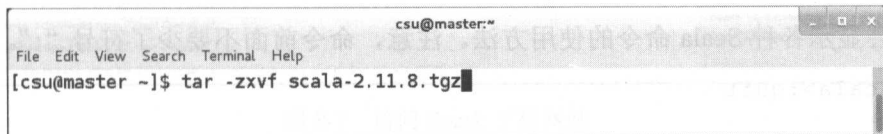


图 8-4 解压缩 Scala 安装包

切换到主安装目录 scala-2.11.8, 执行 “ls -l” 命令, 可以看到 Scala 系统文件目录, 如图 8-5 所示。

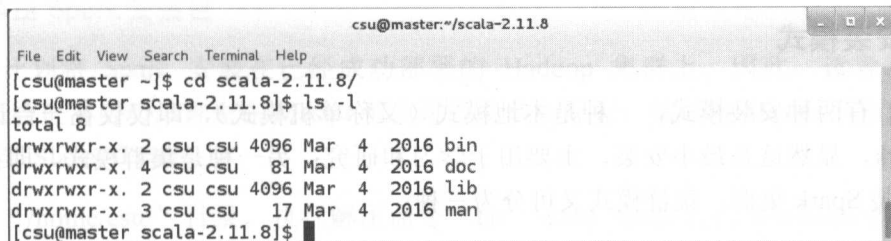


图 8-5 Scala 主安装目录

8.2.3 启动并应用 Scala

进入“/home/csu/scala-2.11.8/”目录，执行“bin/scala”命令，即可启动 Scala，如图 8-6 所示。

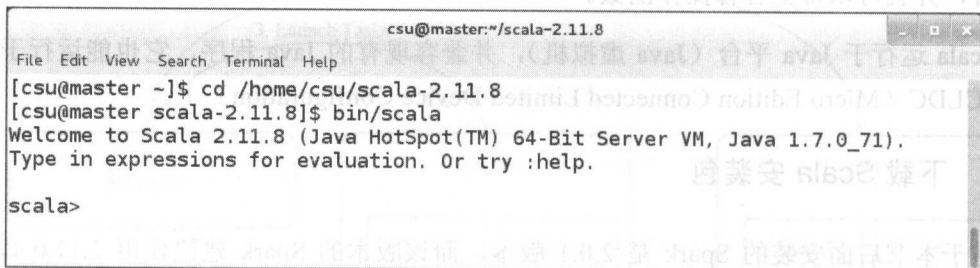


图 8-6 进入 Scala Shell

图 8-6 是 Scala 的 shell，用户可以输入各种 Scala 命令，例如：

```

scala>println("Hello,world")
Hello,world
scala>5*9
res0: Int = 45
scala>2*res0
res1: Int = 90
scala>:help
  
```

这里会显示各种 Scala 命令的使用方法。注意，命令前面不要少了符号“:”。

```
scala>:quit
```

这是退出 scala 的命令。注意，一定要正确退出 Scala 系统，否则有可能导致数据丢失。

8.3 Spark 集群的安装与配置

8.3.1 安装模式

Spark 有两种安装模式，一种是本地模式（又称单机模式），即仅仅在一台计算机上安装 Spark，显然这是最小安装，主要用于学习和研究；另一种是集群模式，即在 Linux 集群上安装 Spark 集群。集群模式又可分为三种：

Standalone 模式：又称为独立部署模式，该模式下系统采用 Spark 自带的简单集群管理器，不依赖第三方提供的集群管理器。这种部署模式比较方便快捷。

Hadoop Yarn 模式：这时候系统利用 Hadoop2.0 以上版本中的 Yarn 充当资源管理器。本书采用了这种模式，因此，下面的安装需要确保 Hadoop2.6 已经安装好，并且在启动 Spark 或在提交 Spark 程序的时候，Hadoop2.6 已经正常启动。

Apache Mesos 模式：Apache Mesos 是由加州大学伯克利分校的 AMPLab 开发的通用群集管理器，支持 Hadoop、ElasticSearch、Spark、Storm 和 Kafka 等系统。Mesos 内核运行在每个机器上，在整个数据中心和云环境内向分布式系统（如 Spark 等）提供资源管理的 API 接口。

8.3.2 Spark 的安装

下面开始安装 Spark。我们这里采用 Hadoop Yarn 模式安装 Spark。

1. 下载 Spark 安装包

读者可以从 <http://spark.apache.org/downloads.html> 下载最新版本的 Spark。图 8-7 是下载 Spark 的网站的首页。

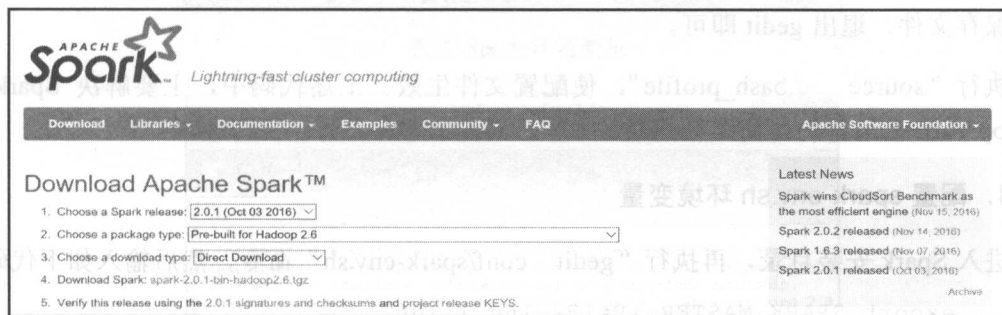


图 8-7 访问 Spark 下载网站

我们选择 `spark-2.0.1-bin-hadoop2.6.tar` 下载。由于 Spark 是一个较为庞大的系统，因此下载时间相对较长。读者也可以在本书第 8 章软件资源目录中找到 `spark-2.0.1-bin-hadoop2.6.tar`，并将其复制到“/home/csu/”下。

2. 解压 Spark 安装包

我们计划将 Spark 安装在已经成功部署的 Hadoop 集群上，因此，读者需要确保 Hadoop 已经正常启动。首先在 Master 节点进行安装和配置，然后将 Master 节点的安装主目录复制到 Slave 节点上。

进入“/home/csu/”目录，执行解压命令“`tar -zxvf spark-2.0.1-bin-hadoop2.6.tar`”，系统生成 Spark 主安装目录 `spark-2.0.1-bin-hadoop2.6`。用户可以进入该目录查看一下 Spark 的系统文件。

3. 配置 Linux 环境变量

执行“`gedit ~/.bash_profile`”命令，如图 8-8 所示。

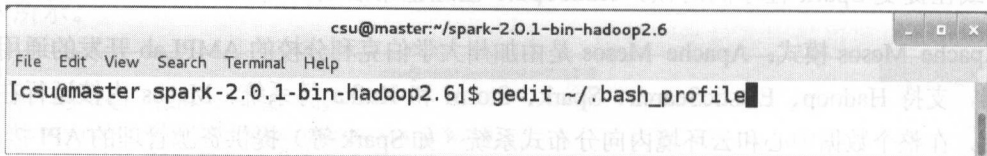


图 8-8 为 Spark 配置 Linux 环境变量

在打开的文件中，增加下面的代码：

```
#spark
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
export HDFS_CONF_DIR=$HADOOP_HOME/etc/hadoop
export YARN_CONF_DIR=$HADOOP_HOME/etc/hadoop
```

保存文件，退出 `gedit` 即可。

执行“`source ~/.bash_profile`”，使配置文件生效。上述代码中，主要解决 Spark 在 Hadoop Yarn 上运行的访问路径问题。

4. 配置 spark-env.sh 环境变量

进入 Spark 安装目录，再执行“`gedit conf/spark-env.sh`”命令，然后输入如下代码：

```
export SPARK_MASTER_IP=192.168.1.100
export JAVA_HOME=/usr/java/jdk1.7.0_71/
export SCALA_HOME=/home/csu/scala-2.11.8/
```

编辑结果如图 8-9 所示。

保存文件，退出编辑器即可。

5. 修改 Spark 的 slaves 文件

执行“`gedit cong/slaves`”命令，在编辑区输入如下代码：

```
slave0
slave1
```

输入情况如图 8-10 所示。

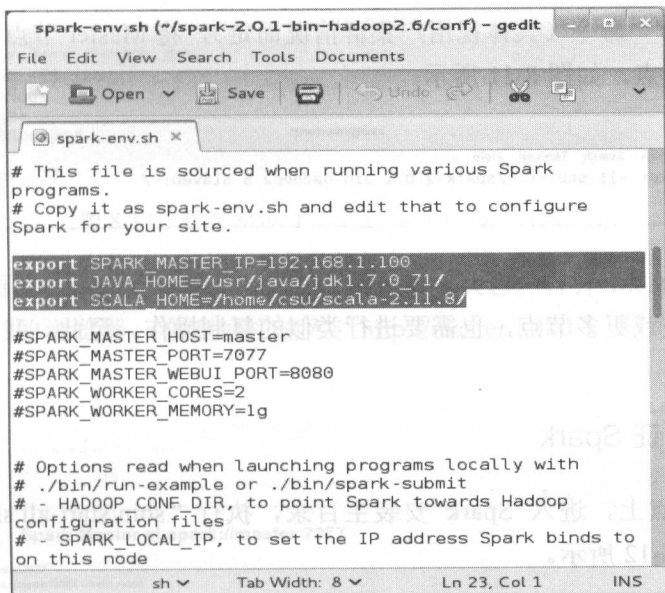


图 8-9 配置 Spark 环境变量 (一)

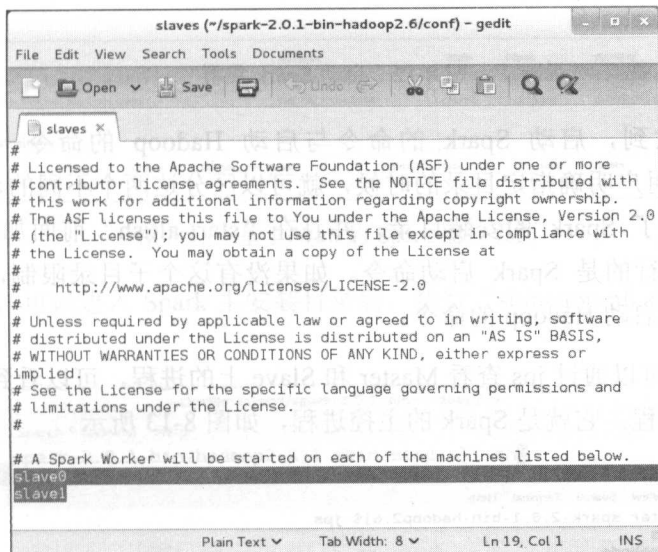


图 8-10 配置 Spark 环境变量(二)

保存文件，退出编辑器即可。

6. 将安装好的 Spark 复制到 Slave 节点

执行 “`scp -r ~/.spark-2.0.1-bin-hadoop2.6 slave0:~/`” 和 “`scp -r ~/.spark-2.0.1-bin-`

hadoop2.6 slave1:~/”命令（具体视用户集群情况而定），将 Master 节点上的 Spark 系统文件复制到 Slave0 节点，如图 8-11 所示。

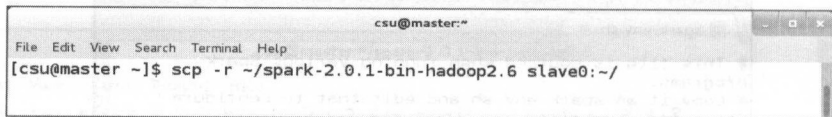


图 8-11 将 Spark 主安装目录复制到 Slave0 节点

如果有 Slave1 或更多节点，也需要进行类似的复制操作。至此，我们就完成了 Spark 的安装。

8.3.3 启动并验证 Spark

在 Master 节点上，进入 Spark 安装主目录，执行“sbin/start-all.sh”命令即可启动 Spark 集群，如图 8-12 所示。

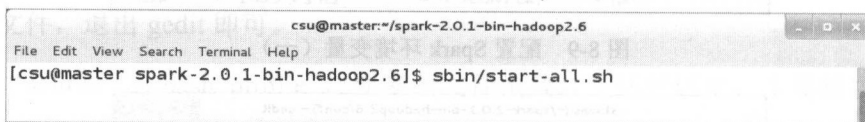


图 8-12 启动 Spark 集群

读者可能注意到，启动 Spark 的命令与启动 Hadoop 的命令一样，都是“start-all.sh”。但是，当用户明确指定目录的时候，就可以区分这两个不同的命令了。显然由于我们这里已经进入了 Spark 的安装目录，并且在“start-all.sh”前面加上了 sbin 子目录名，这就确保了执行的是 Spark 启动命令。如果没有这个子目录限制，而是简单地使用“start-all.sh”，则是启动 Hadoop 的命令。

启动后，我们可以通过 jps 查看 Master 和 Slave 上的进程，可以看到，在 Master 上增加了一个 Master 进程，它就是 Spark 的主控进程，如图 8-13 所示。

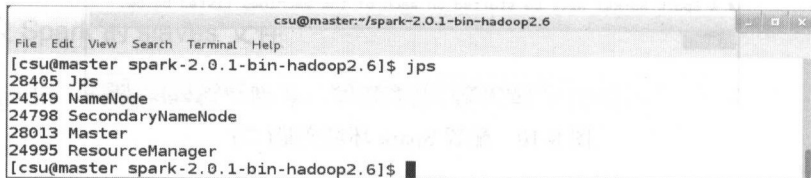


图 8-13 查看 Spark 进程

在 Slave 节点（如 Slave0）上执行 jps 命令，则可以看到增加了 Spark 的 Worker 进程，如图 8-14 所示。

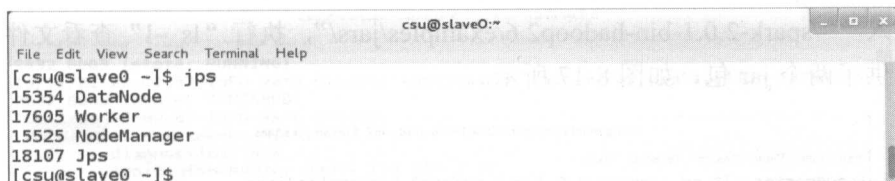


图 8-14 查看 Slave0 节点上的 Spark worker 进程

我们也可以通过 Spark 提供的 Web 接口查看系统状态。打开 Master 节点（也可以是任何其他节点）上的浏览器，在地址栏输入“http://master:8080”，即可看到如图 8-15 所示的监控画面。

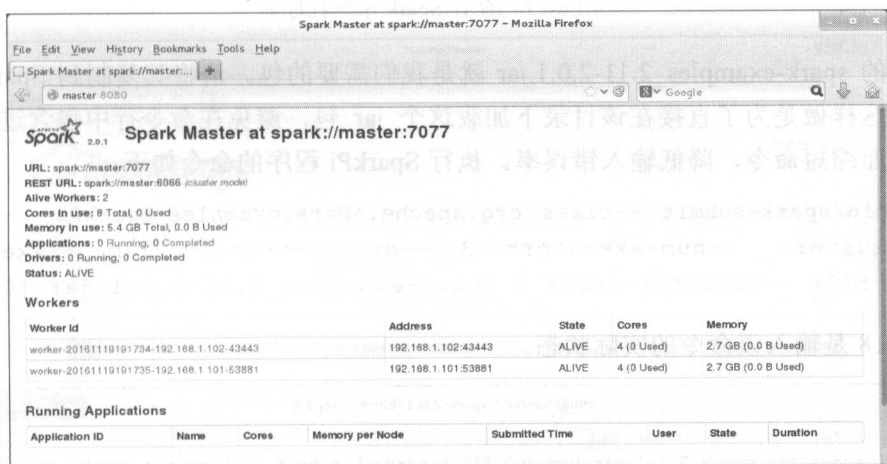


图 8-15 基于 Web 的 Spark 监控页面

要退出 Spark，可以进入 Spark 主安装目录后，执行“sbin/stop-all.sh”命令，如图 8-16 所示。

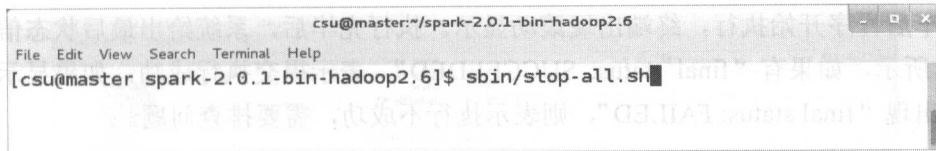


图 8-16 关闭 Spark 集群

验证 Spark 是否安装成功的另一个有效方法，就是通过终端命令行向 Spark 集群提交计算程序，SparkPi 就是最常用的验证程序，相当于 Spark 的“Hello, world”。

为了使读者能够顺利执行 SparkPi 程序，我们建议读者首先浏览一下 Spark 主安装目录下的“examples/jars/”子目录中的示例程序文件。

请进入 “~/spark-2.0.1-bin-hadoop2.6/examples/jars/”，执行 “ls -l” 查看文件，可以看到系统提供了两个 jar 包，如图 8-17 所示。

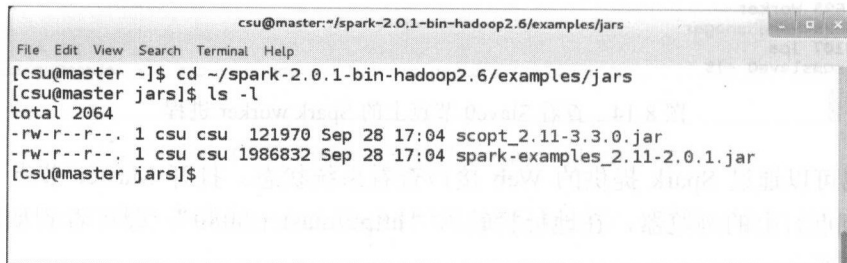


图 8-17 查看 Spark 示例程序

图中的 spark-examples_2.11-2.0.1.jar 就是我们需要的包。请将其复制到 Spark 主安装目录下，这样做是为了直接在该目录下加载这个 jar 包，避免在命令行中包含过多的路径参数，从而缩短命令，降低输入错误率。执行 SparkPi 程序的命令如下：

```
bin/spark-submit --class org.apache.spark.examples.SparkPi --master
yarn-cluster --num-executors 3 --driver-memory 512m --executor-
memory 512m --executor-cores 1 spark-examples_2.11-2.0.1.jar 10
```

图 8-18 是输入该命令的实际状态。

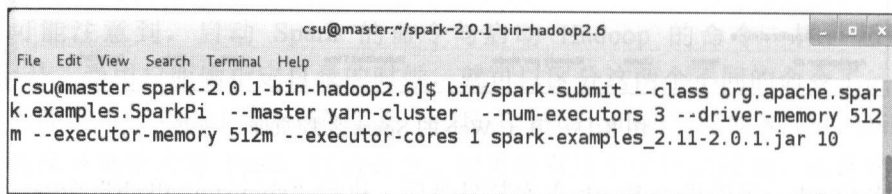


图 8-18 准备执行 Pi 程序

回车后程序开始执行，终端出现滚动显示。执行完毕后，系统给出最后状态信息，如图 8-19 所示，如果有 “final status: SUCCEEDED”，表示提交执行成功；如果显示的状态信息中出现 “final status: FAILED”，则表示执行不成功，需要排查问题。

要查看执行结果，读者可以打开图 8-19 中的跟踪 URL (tracking URL，注意观察图 8-19)：

```
http://master:18088/proxy/application_1479864689832_0003/A
```

实际上，只要将鼠标移到该链接上，单击右键，选择 “open link”，即可通过 Firefox 浏览器打开需要的页面，如图 8-20 所示。

```

csu@master:~/spark-2.0.1-bin-hadoop2.6
File Edit View Search Terminal Help
9832_0003 (state: RUNNING)
16/11/22 23:02:37 INFO yarn.Client: Application report for application_147986468
9832_0003 (state: FINISHED)
16/11/22 23:02:37 INFO yarn.Client:
  client token: N/A
  diagnostics: N/A
  ApplicationMaster host: 192.168.1.101
  ApplicationMaster RPC port: 0
  queue: default
  start time: 1479884542469
  final status: SUCCEEDED
  tracking URL: http://master:18088/proxy/application_1479864689832_0003/
A
  user: csu
16/11/22 23:02:37 INFO util.ShutdownHookManager: Shutdown hook called
16/11/22 23:02:37 INFO util.ShutdownHookManager: Deleting directory /tmp/spark-4
6b5c3b2-cb12-4e4e-9738-344c690614c7
[csu@master spark-2.0.1-bin-hadoop2.6]$

```

图 8-19 Pi 程序执行成功后的显示

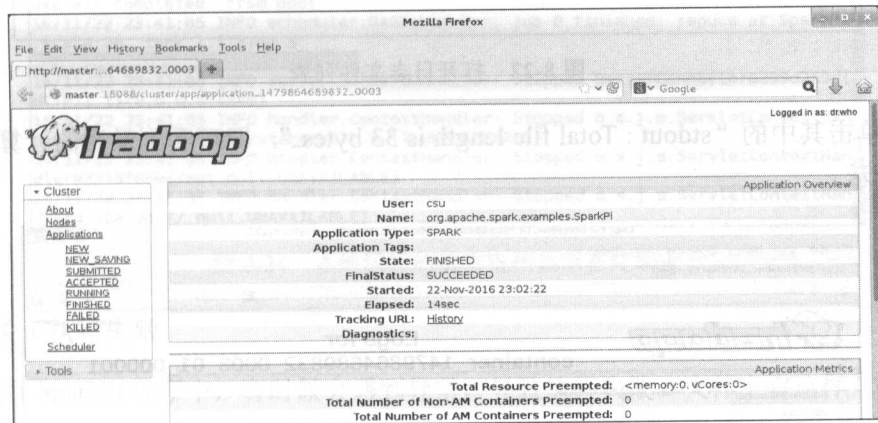


图 8-20 程序执行跟踪页面

在图 8-20 中，将页面往下滚动一点，可以看到在 ApplicationMaster 栏下，有“Start Time”、“slave0:8042”和“logs”的显示内容，如图 8-21 所示。

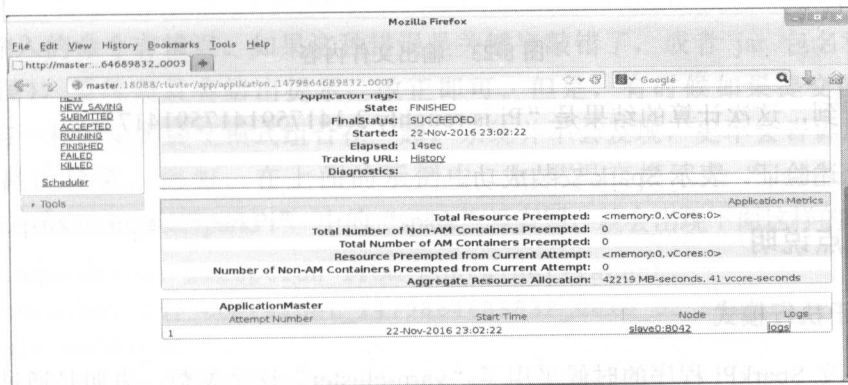


图 8-21 日志文件的位置

单击右边的“logs”，进入如图 8-22 所示的页面。

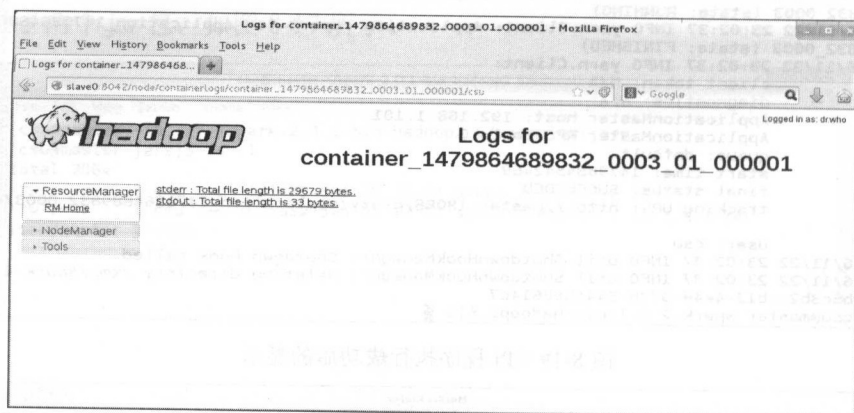


图 8-22 打开日志文件列表

然后单击其中的“stdout: Total file length is 33 bytes.”，即可看到最后的计算结果，如图 8-23 所示。

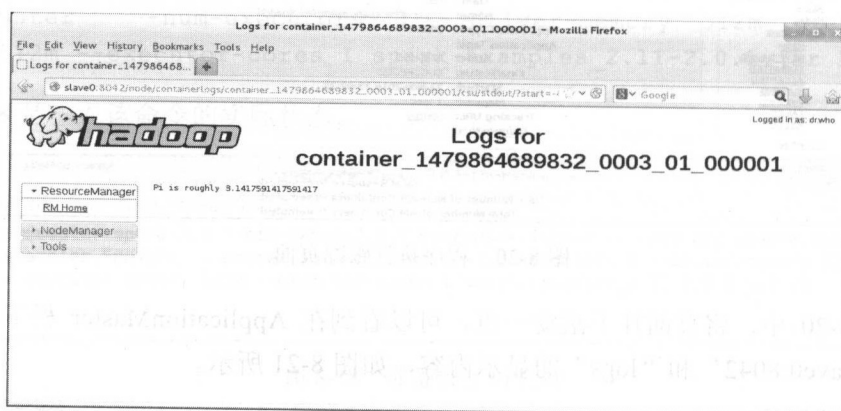


图 8-23 输出文件内容

可以看到，这次计算的结果是“Pi is roughly 3.1417591417591417”。

完成上述验证，表示 Spark 安装成功。

8.3.4 几点说明

1. 关于执行模式

上面提交 SparkPi 程序的时候采用了“yarn-cluster”这个参数，表明是通过 Yarn 集群

模式执行该程序的。实际上，我们可以有三种执行模式。除了已经验证的 Yarn 集群模式外，还有本地模式和终端模式。例如，用“local”代替“yarn-cluster”，即表示通过本地模式执行程序；用“yarn-client”代替“yarn-cluster”，则表示通过终端模式执行程序。如果系统安装和配置成功，也都能顺利完成任务。后面两种执行的结果都会显示在终端上，图 8-24 给出了本地模式执行的结果。

```

File Edit View Search Terminal Help
(TID 2) in 1332 ms on slave1 (9/10)
16/11/22 23:41:05 INFO scheduler.TaskSetManager: Finished task 1.0 in stage 0.0
(TID 1) in 1349 ms on slave1 (10/10)
16/11/22 23:41:05 INFO scheduler.DAGScheduler: ResultStage 0 (reduce at SparkPi.
scala:38) finished in 1.379 s
16/11/22 23:41:05 INFO cluster.YarnScheduler: Removed TaskSet 0.0, whose tasks h
ave all completed, from pool
16/11/22 23:41:05 INFO scheduler.DAGScheduler: Job 0 finished: reduce at SparkPi
.scala:38, took 1.718760 s
Pi is roughly 3.1405711405711405
16/11/22 23:41:05 INFO server.ServerConnector: Stopped ServerConnector@289d259d{
HTTP/1.1}{0.0.0.0:4040}
16/11/22 23:41:05 INFO handler.ContextHandler: Stopped o.s.j.s.ServletContextHan
dler@3652831e{/stages/stage/kill,null,UNAVAILABLE}
16/11/22 23:41:05 INFO handler.ContextHandler: Stopped o.s.j.s.ServletContextHan
dler@3345f6de{/api,null,UNAVAILABLE}
16/11/22 23:41:05 INFO handler.ContextHandler: Stopped o.s.j.s.ServletContextHan
dler@718a5e8e{/,null,UNAVAILABLE}

```

图 8-24 本地运行的结果直接显示在终端上

2. 关于执行失败

在执行 SparkPi 程序中，有时候会遇到异常，导致执行失败。大体上有这样几个原因：

一是采用集群模式（包括 Client 模式和 Cluster 模式）提交的时候，Yarn 没有启动，因此这时候没有资源管理器介入。这种情况系统会给出提示，解决办法自然是启动 Yarn。实际上，启动 2.0 以上版本的 Hadoop 的时候，其自带的 Yarn 也会一并启动，所以我们这里只要确保已经启动了 Hadoop 即可。

二是输入的命令有错误。如果这种错误是关键字敲错了，或者 jar 包名称错误（包括路径错误），系统尚且会给出提示，改正即可。但是，有时候如果提交的类名有错误，并且与一种特定的提交模式组合在一起，系统并不会发现，更不会告诉用户错误原因，而是抛出异常。例如，在上面的示例中，如果提交模式是“yarn-cluster”，且“org.apache.spark.examples.SparkPi”中的“spark”没有写，就会出现下面这样的异常信息：

```

Diagnostics: Exception from container-launch.
Container id: container_1479864689832_0010_02_000001
Exit code: 10
Stack trace: ExitCodeException exitCode=10:

```

这时候用户往往会陷入异常代码分析而无法找到症结，所以，这时候我们要仔细检查自己输入的命令是否正确。可见，应用经验有时候非常有用。

8.4 开发环境安装与配置

8.4.1 IDEA 简介

IDEA 是一个通用的集成开发环境，Spark 通常采用 Scala 语言进行开发，而 IDEA 则是最佳的 Scala 语言开发环境。

8.4.2 IDEA 的安装

下载 IDEA 的官方网站是 <https://www.jetbrains.com/idea/download/#section=linux>，如图 8-25 所示，我们选择了最新的 IDEA 安装文件 `ideaIC-2016.3.tar.gz`。读者也可以在本书第 8 章的软件资源目录中找到 `ideaIC-2016.3.tar.gz` 文件。



图 8-25 下载 IntelliJ IDEA 的官方网站

请将 `ideaIC-2016.3.tar.gz` 文件复制到 Master 节点的 `/home/csu/` 目录下，然后执行解压缩命令 `“tar -zxvf ideaIC-2016.3.tar.gz”`，如图 8-26 所示。

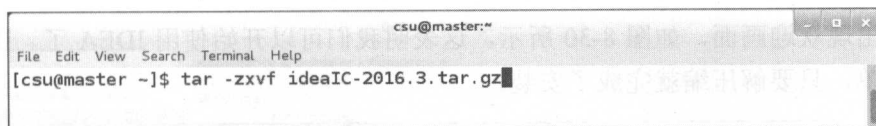


图 8-26 解压缩 IntelliJ IDEA 安装包

回车后系统开始解压缩，等待片刻后，系统创建安装目录 `idea-IC-163.7743.44`。进入该目录，执行“`ls -l`”可以查看 IDEA 的系统文件和子目录，如图 8-27 所示。

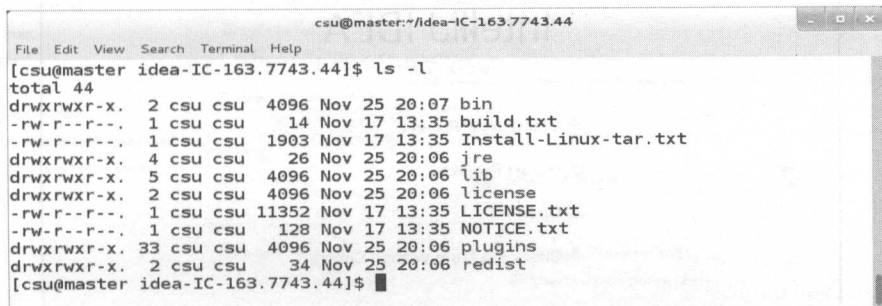


图 8-27 IDEA 的安装目录

在 `idea-IC-163.7743.44` 目录下，执行“`bin/idea.sh`”，开始运行 IDEA，如图 8-28 所示。

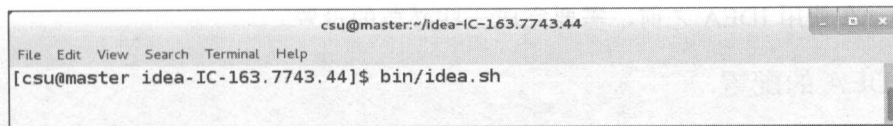


图 8-28 启动 IDEA

首先出现的是许可协议申明，如图 8-29 所示，单击“Accept”按钮即可。

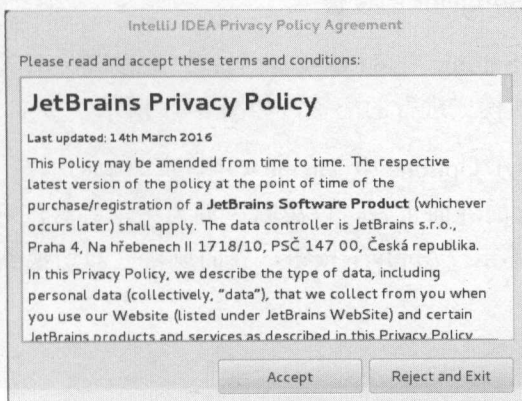


图 8-29 首次启动 IDEA 时候的协议许可

之后出现欢迎画面,如图 8-30 所示,这表明我们可以开始使用 IDEA 了。IDEA 的安装非常简单,只要解压缩就完成了安装。

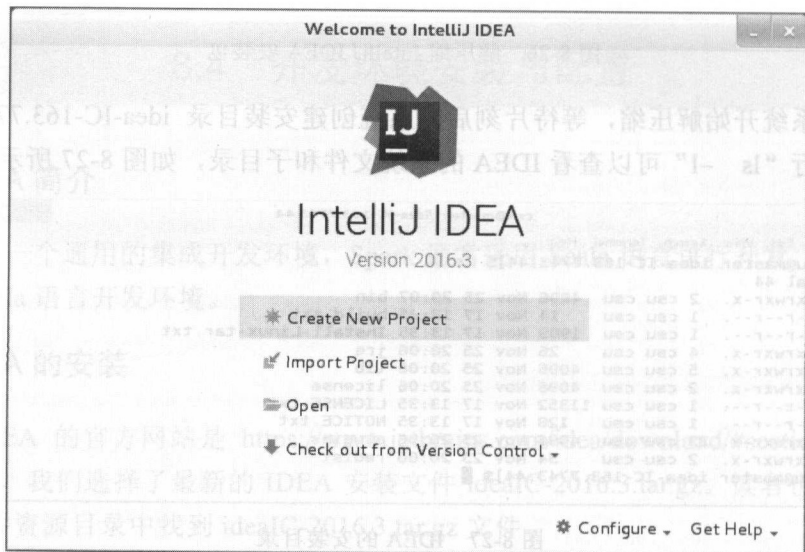


图 8-30 IDEA 启动后的欢迎页面

但是,在使用 IDEA 之前,需要完成一些必要的设置。

8.4.3 IDEA 的配置

在图 8-30 中,单击右下方的 Configure 的下拉式列表,在弹出的菜单中选择 Setting,开始对 IDEA 环境进行设置。

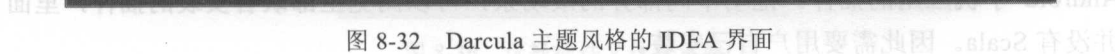
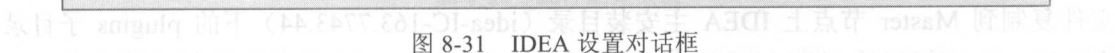
1. Appearance & Behavior 的设置

在弹出的“Default Settings”窗口中,我们首先来设置一下 Appearance & Behavior,这是对 IDEA 外观和表现模式进行的配置,如图 8-31 所示。

这里仅仅修改一下 UI Options 中 Theme (主题),例如选择“Darcula”风格的主题。Darcula 是一种十分流行的界面主题,以黑暗色调为主,具有沉静稳健的特点,深受广大用户喜欢。选择完毕,单击“Apply”按钮,界面风格立即切换到 Darcula 主题风格,如图 8-32 所示。

2. 安装 Scala 插件

为了在 IDEA 中开发 Scala 程序,需要为 IDEA 安装 Scala 插件。



可以访问 <http://plugins.jetbrains.com/> 下载 Scala 插件安装压缩包 `scala-intellij-bin-2016.3.3.zip` 文件, 图 8-33 是其主页面画面, 在该页下面一点的地方可以找到下载 Scala 插件的链接。



图 8-33 下载 IDEA 插件的官方网站

读者也可以在本书第 8 章的软件资源目录中找到 `scala-intellij-bin-2016.3.3.zip` 文件。

为了在 IDEA 集成开发环境中安装 Scala 插件, 请将 `scala-intellij-bin-2016.3.3.zip` 文件复制到 Master 节点上 IDEA 主安装目录 (`idea-IC-163.7743.44`) 下的 `plugins` 子目录中, 图 8-34 是复制完成后通过文件查看命令显示的文件列表, 可以看到 `scala-intellij-bin-2016.3.3.zip` 已经在里面。

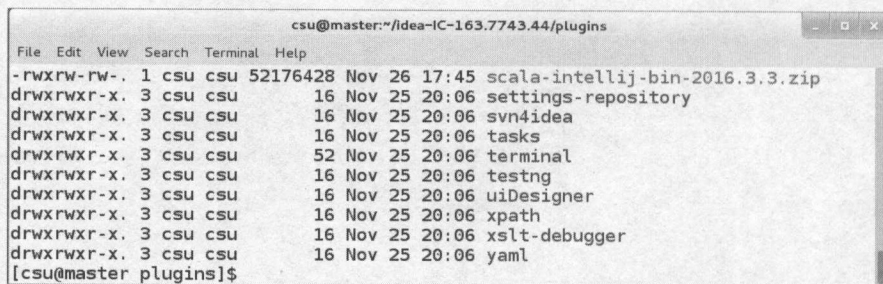


图 8-34 将 `scala-intellij-bin-2016.3.3.zip` 复制到 `plugins` 子目录中

准备好 Scala 插件安装压缩包后, 就可以开始安装了。

回到 IDEA 的“Default Settings”页面, 单击左边的“Plugins”, 系统切换到插件配置页面, 如图 8-35 所示。

可以看到, 系统已经安装了很多插件, 例如“Android Support”, 这是用于开发 Android 手机应用的插件。拖动中间部分的滚动条, 可以浏览全部缺省安装的插件, 里面并没有 Scala。因此需要用户自己安装。

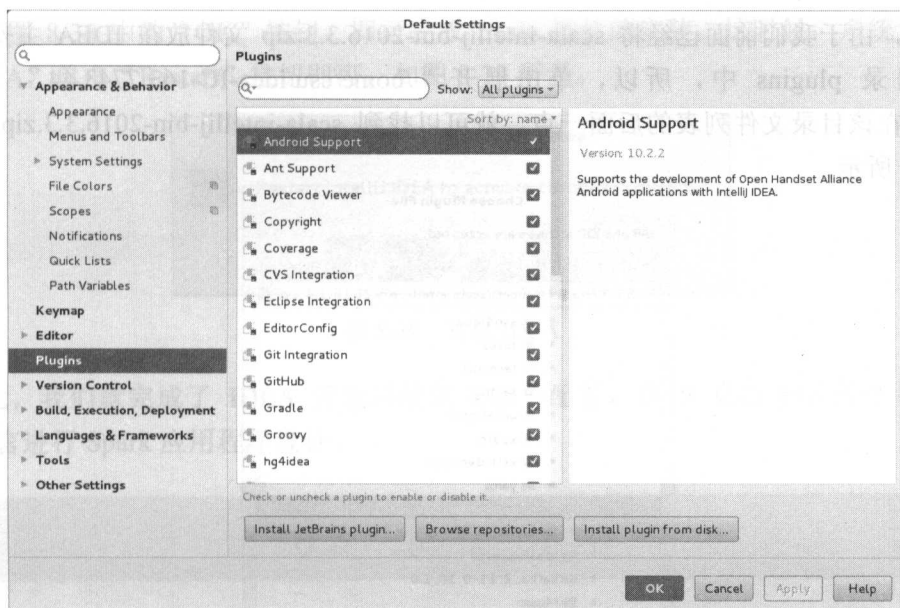


图 8-35 插件配置页面

单击图 8-35 右下边的“Install plugin from disk...”，出现如图 8-36 的对话框。在该页面中，用户可以从任意目录中选择 Scala 插件安装压缩包或 jar 包（注意窗口上方有一行文字“JAR and ZIP archives are accepted”）。

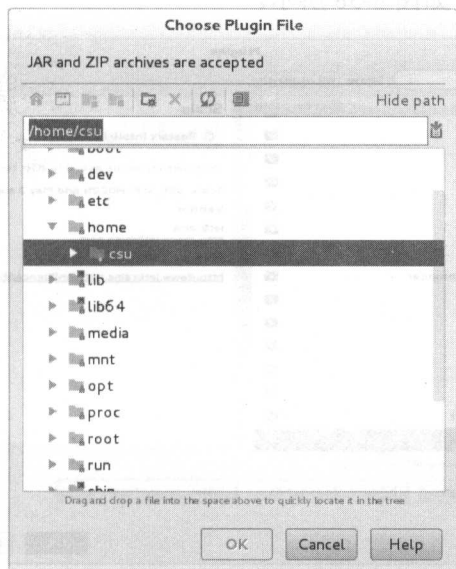


图 8-36 选择插件文件对话框

显然，由于我们前面已经将 `scala-intellij-bin-2016.3.3.zip` 文件放在 IDEA 主安装目录下的子目录 `plugins` 中，所以，单击展开 “`/home/csu/idea-IC-163.7743.44`”，一直到 `plugins`，在该目录文件列表的后面一点，就可以找到 `scala-intellij-bin-2016.3.3.zip` 文件，如图 8-37 所示。

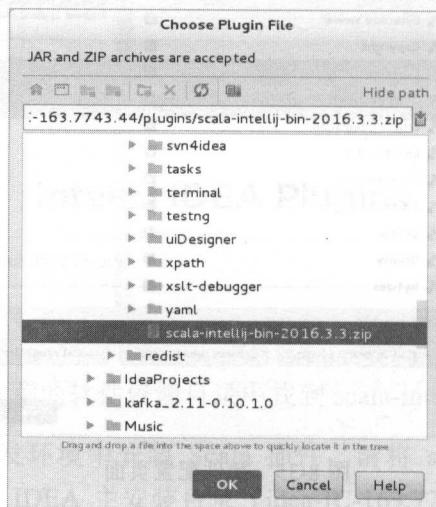


图 8-37 定位到 `scala-intellij-bin-2016.3.3.zip` 文件

选定 Scala 插件安装压缩包后，单击图 8-37 中的 “OK” 按钮，系统立即将 Scala 插件信息显示在插件列表中，如图 8-38 所示。

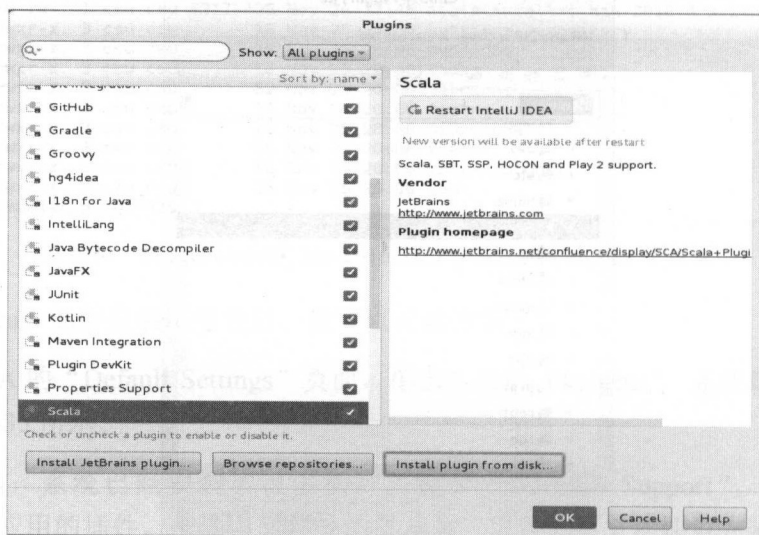


图 8-38 系统安装的 Scalal 插件

单击图 8-38 中的“OK”按钮，即可完成 Scala 插件的安装。这时候，系统询问是否重启 IDEA，单击“Restart”按钮即可，如图 8-39 所示。

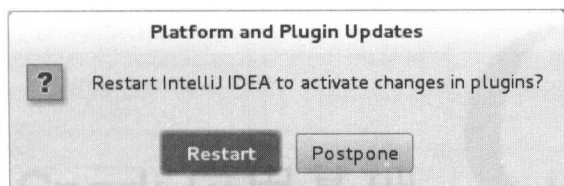


图 8-39 重启 IDEA

至此，我们就完成了 IDEA 开发环境的安装与配置，第 9 章将介绍利用 IDEA 和 Scala 语言进行 Spark 应用程序设计。

8.5 本章小结

本章首先介绍了 Spark 架构及其工作原理，然后详细介绍了 Spark 及其开发环境的安装与配置。

第 8 章我们介绍了 Spark 的架构与关键概念，并搭建了 Spark 开发环境。本章我们要在搭建的开发平台上，设计并实现两个典型的 Spark 应用程序。由于 Spark 程序主要有两种运行模式，即集群模式和客户端模式，所以，我们首先介绍 Spark 的运行模式。

9.1 Spark 程序的运行模式

9.1.1 Spark on Yarn-cluster

1. Spark 和 Yarn 的配合关系

Spark 的计算调度方式，可以通过 Yarn 模式实现。Spark 利用 Yarn 集群提供的丰富计算资源，将任务分布式的运行起来。Spark on Yarn 分两种模式：Yarn-cluster 和 Yarn-client。

Yarn-cluster 模式的运行框架如图 9-1 所示。

2. Spark on Yarn-cluster 实现流程

- (1) 由客户端生成 Application 信息，提交给 ResourceManager。
- (2) ResourceManager 为 Spark Application 分配第一个 Container，并在指定节点的 Container 上启动 SparkContext。
- (3) SparkContext 向 ResourceManager 申请资源以运行 Executor。ResourceManager 分配 Container 给 SparkContext，SparkContext 和相关的 NodeManager 通信，在获得的

Container 上启动 CoarseGrainedExecutorBackend, CoarseGrainedExecutorBackend 启动后, 开始向 SparkContext 注册并申请 Task。

- (4) SparkContext 分配 Task 给 CoarseGrainedExecutorBackend 执行。
- (5) CoarseGrainedExecutorBackend 执行 Task 并向 SparkContext 汇报运行状况。

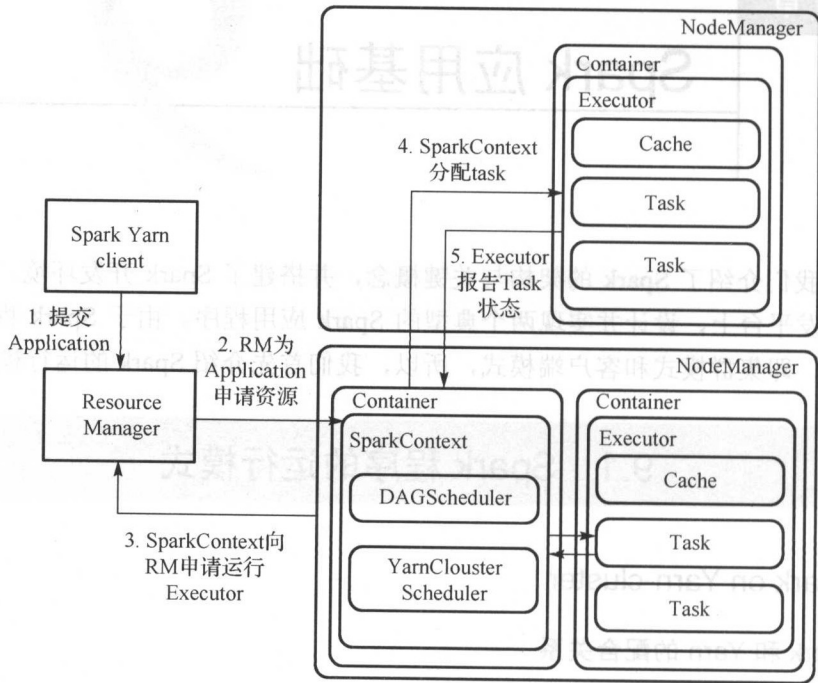


图 9-1 Spark on Yarn-cluster 运行框架

9.1.2 Spark on Yarn-client

Yarn-client 模式的运行框架如图 9-2 所示。

(1) 客户端向 ResourceManager 发送 Spark 应用提交请求, ResourceManager 为其返回应答, 该应答中包含多种信息 (如 ApplicationId、可用资源使用上限和下限等)。Client 端将启动 ApplicationMaster 所需的所有信息打包, 提交给 ResourceManager。

(2) ResourceManager 收到请求后, 会为 ApplicationMaster 寻找合适的节点, 并在该节点上启动它。ApplicationMaster 是 Yarn 中的角色, 在 Spark 中进程名字是 ExecutorLauncher。

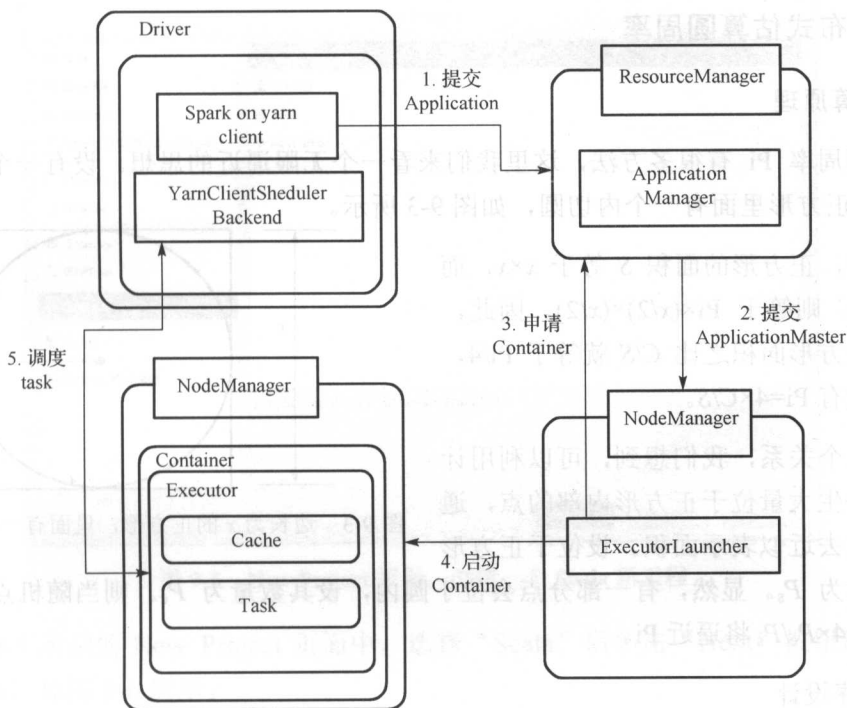


图 9-2 Spark on Yarn-client 运行框架

(3) 根据每个任务的资源需求, ApplicationMaster 可向 ResourceManager 申请一系列用于运行任务的 Container。

(4) 当 ApplicationMaster (从 ResourceManager 端) 收到新分配的 Container 列表后, 会向对应的 NodeManager 发送信息以启动 Container。ResourceManager 分配 Container 给 SparkContext, SparkContext 和相关的 NodeManager 通信, 在获得的 Container 上启动 CoarseGrainedExecutorBackend, CoarseGrainedExecutorBackend 启动后, 开始向 SparkContext 注册并申请 Task。

(5) SparkContext 分配 Task 给 CoarseGrainedExecutorBackend 执行, CoarseGrainedExecutorBackend 执行 Task 并向 SparkContext 汇报运行状况。

9.2 Spark 应用设计

由于 Spark 是基于内存的迭代计算框架, 因此, 非常适用于需要多次操作特定数据集的应用场合。本节介绍两个典型应用。

9.2.1 分布式估算圆周率

1. 计算原理

计算圆周率 π 有很多方法，这里我们来看一个无限逼近的思想。设有一个正方形，边长是 x ，正方形里面有一个内切圆，如图 9-3 所示。

上图中，正方形的面积 S 等于 $x \times x$ ，而圆的面积 C 则等于 $\pi \times (x/2) \times (x/2)$ 。因此，圆面积与正方形面积之比 C/S 就等于 $\pi/4$ ，于是，我们有 $\pi = 4 \times C/S$ 。

根据这个关系，我们想到，可以利用计算机随机产生大量位于正方形内部的点，通过点的数量去近似表示面积。设位于正方形中点的数量为 P_s 。显然，有一部分点会位于圆内，设其数量为 P_c 。则当随机点的数量趋近无穷时， $4 \times P_c/P_s$ 将逼近 π 。

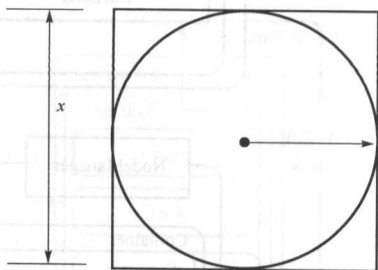


图 9-3 边长为 x 的正方形，里面有一个内切圆

2. 程序设计

启动 IDEA，进入欢迎画面，如图 9-4 所示。

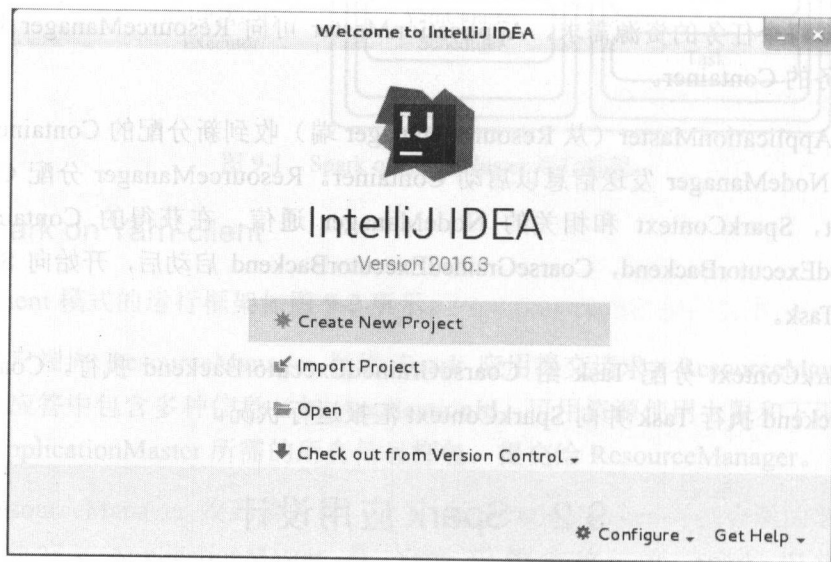


图 9-4 启动 IDEA 后进入欢迎画面

在图 9-4 中选择“Create New Project”，创建一个 Scala 新工程，如图 9-5 所示。

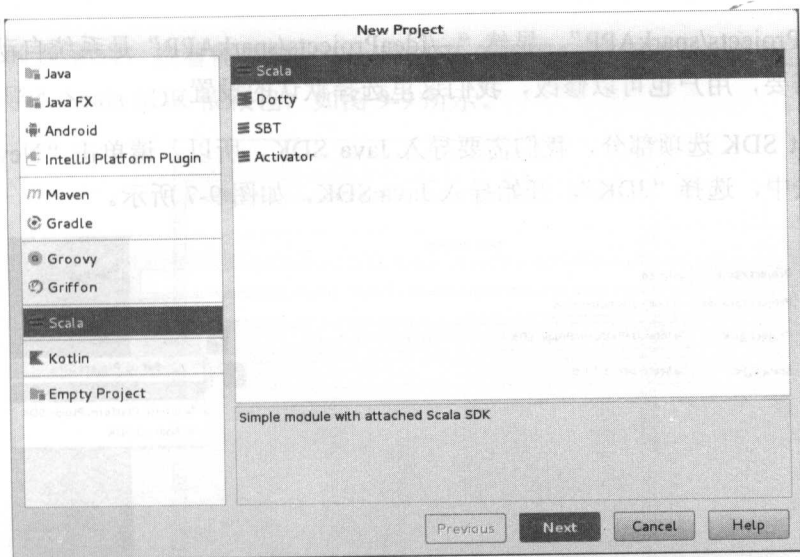


图 9-5 New Project 页面：创建一个 Scala 新工程

在图 9-5 所示的 New Project 页面中，选择“Scala”后单击“Next”按钮进入工程基本设置页面，如图 9-6 所示。

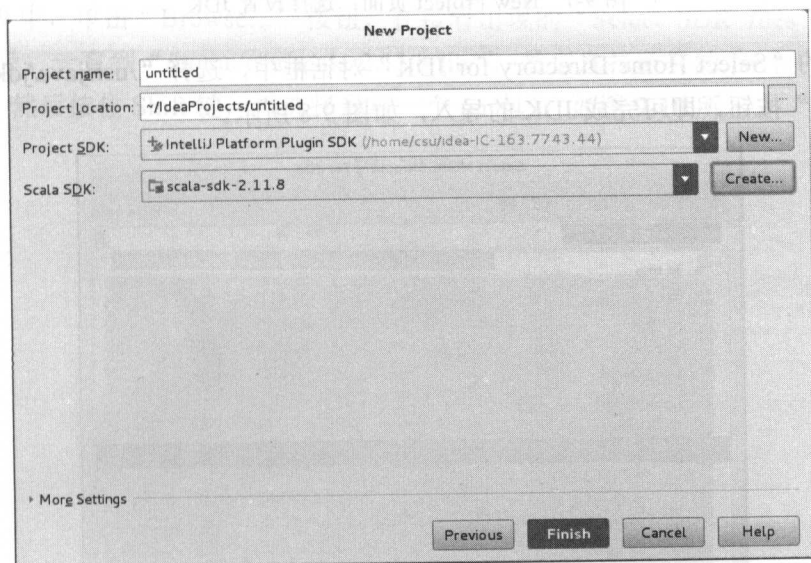


图 9-6 New Project 页面：设置工程名称、位置等信息

在图 9-6 中，开发人员需要设置工程名称、工程位置、Project SDK 和 Scala SDK 等。这里我们给工程取名为 sparkAPP（读者可以任意取一个名称），Project location 被设

置为“~/IdeaProjects/sparkAPP”。显然“~/IdeaProjects/sparkAPP”是系统自动创建的工作目录，如果需要，用户也可以修改，我们这里选择默认的设置。

在 Project SDK 选项部分，我们需要导入 Java SDK。所以，请单击“New...”按钮，在弹出的列表中，选择“JDK”，开始导入 Java SDK，如图 9-7 所示。

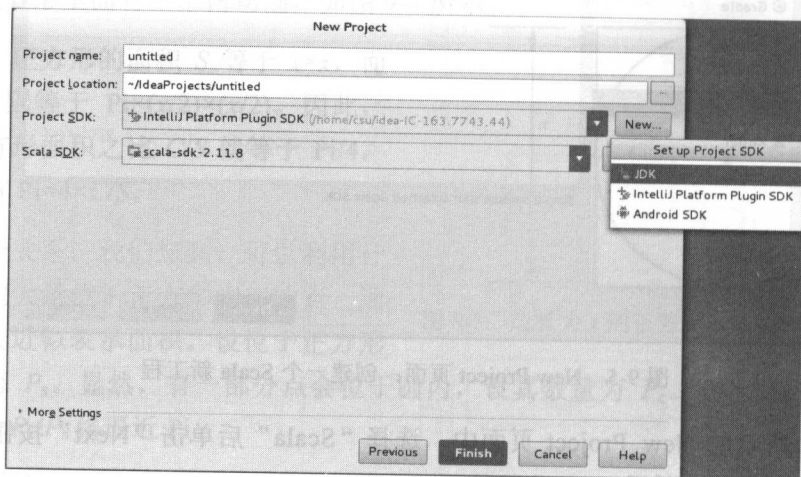


图 9-7 New Project 页面：选择设置 JDK

在弹出的“Select Home Directory for JDK”对话框中，选择“/usr/java/jdk1.7.0_71”，再单击“OK”按钮，即可完成 JDK 的导入，如图 9-8 所示。

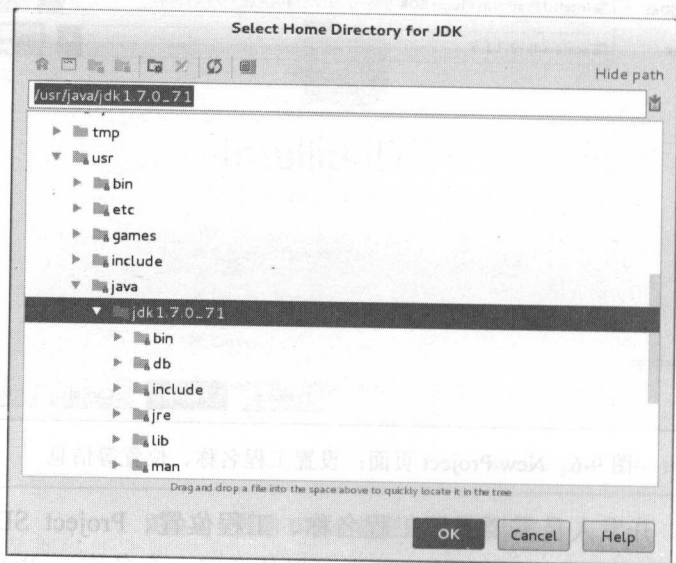


图 9-8 选择已安装的 jdk1.7.0_71

完成 JDK 配置后,接着需要进行 Scala SDK 的配置。单击图 9-7 右侧的“Create...”按钮,开始导入 Scala SDK 依赖包,如图 9-9 所示。

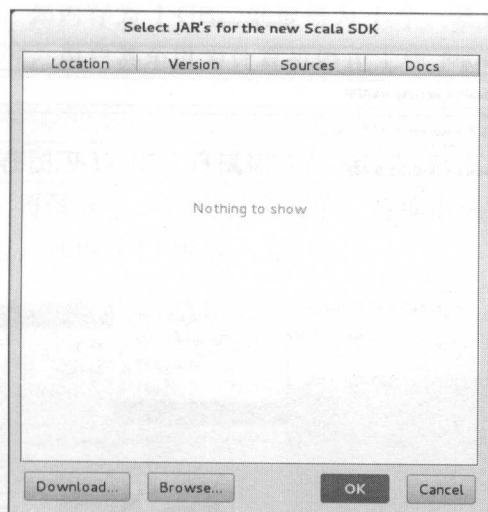


图 9-9 准备选择 Scala SDK 依赖包

在图 9-9 中,单击“Browse...”按钮,在接着出现的“Select SDK files”对话框中,通过展开目录,一直到“/home/csu/scala-2.11.8/lib/”,然后选择所有的 jar 包(首先选择第一个 jar 包,然后按住 Shift 键,再选择最后一个 jar 包),如图 9-10 所示。

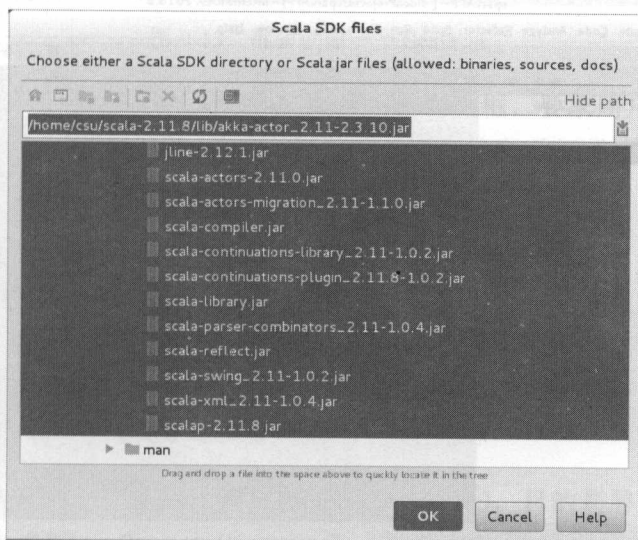


图 9-10 选择/home/csu/ scala-2.11.8/lib 下的所有 jar 包

单击“OK”按钮，返回 New Project 页面，可以看到 Scala-sdk-2.11.8 已经被导入，如图 9-11 所示。

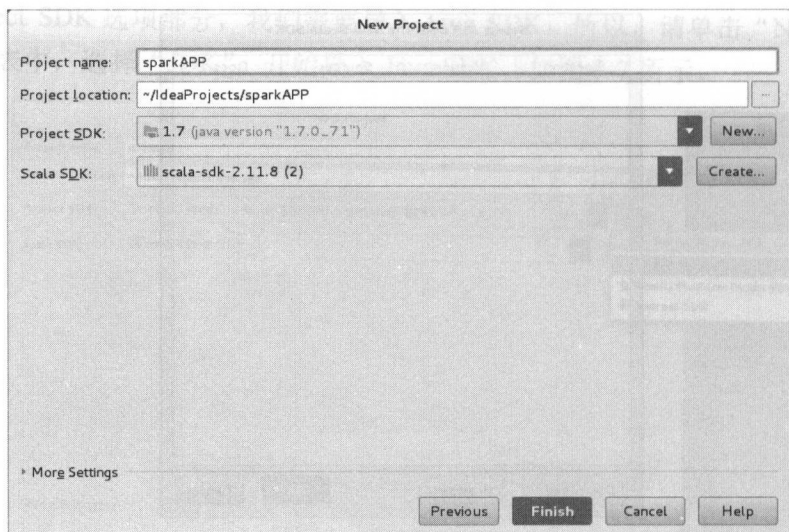


图 9-11 完成 Scala SDK 的配置

单击图 9-11 中的“Finish”按钮，完成新工程设置，系统开始对设置进行索引 (Indexing)，并进入如图 9-12 所示的主界面。

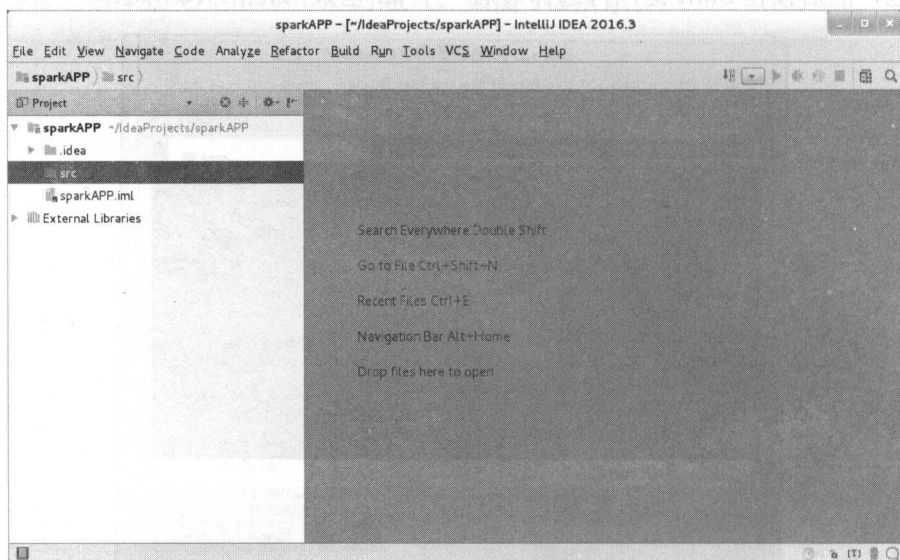


图 9-12 进入开发环境主界面

值得指出的是, IDEA 对新工程的索引需要时间, 长短视虚拟机配置而定。如果配置较好, 例如, 内存为 4 GB, 处理器为 4 个, 硬盘达到 80 GB, 那么索引耗时仅需要几分钟; 如果虚拟机配置较低, 如内存为 1 GB, 处理器为 1 个, 硬盘设置为 20 GB, 那么索引耗时可能会达到几个小时, 用户甚至以为计算机停止工作了。可见系统配置对运行效率的影响是非常大的。

下面开始设计程序。在图 9-12 中, 将鼠标放在 `src` 上, (右键单击鼠标) 展开列表式菜单, 然后选择 “New”, 再展开下一级列表式菜单, 请单击选择 “Package”, 这意味着我们准备创建一个 Scala 包, 如图 9-13 所示。

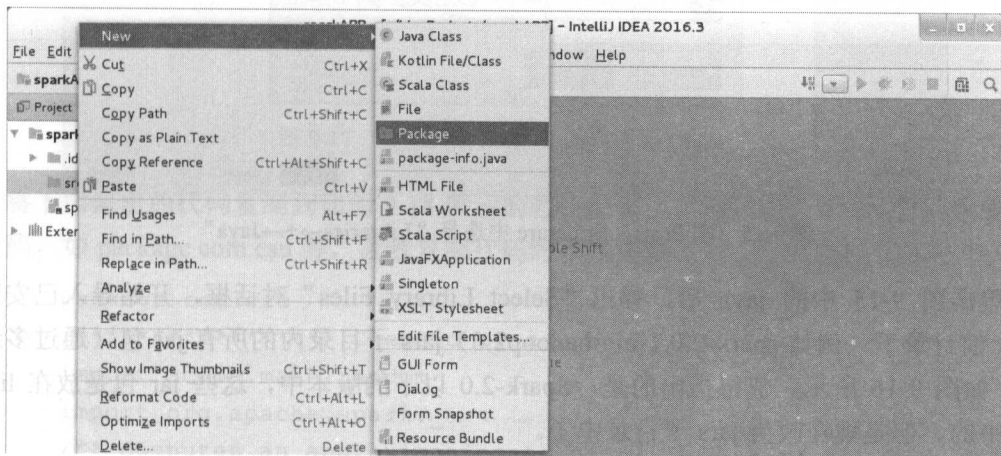


图 9-13 准备创建一个 Scala 包

选择 “Package” 后, 系统弹出如图 9-14 所示的对话框, 要求用户输入新的 Package 名称, 用户可自行命名, 例如输入 `com.csu`, 输入完成后单击 “OK” 按钮。

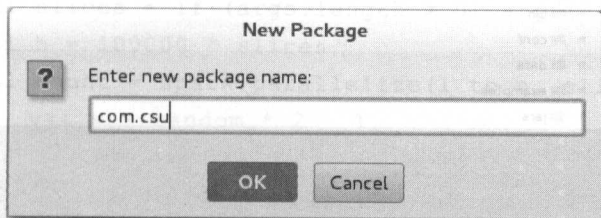


图 9-14 输入新的 Package 名称

接下来, 我们先配置一下工程结构 (Project Structure), 目的是导入 Spark 依赖包。依次选择主界面的 “File→Project Structure”, 在弹出的对话框中选择 “Libraries→+→Java”, 如图 9-15 所示。

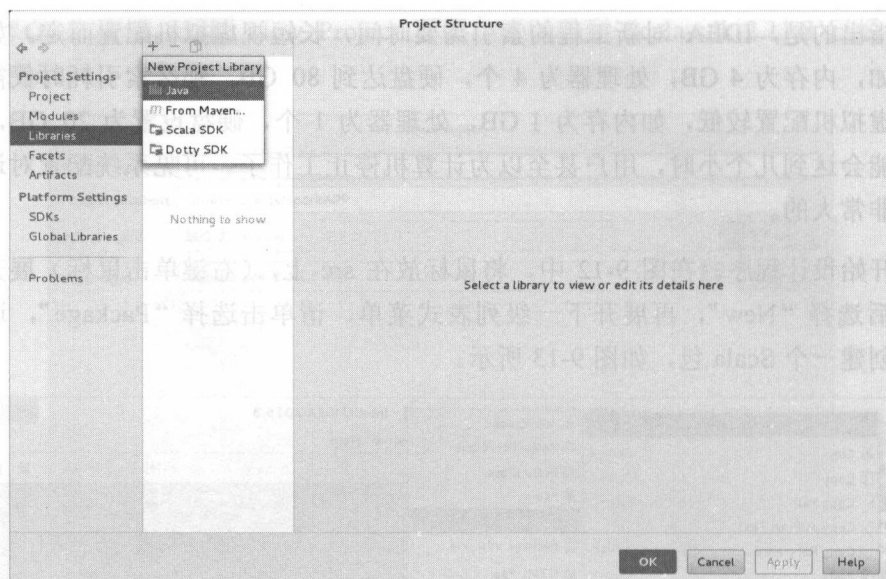


图 9-15 在 Project Structure 中选择“Libraries→+→Java”

单击图 9-15 中的 java 后，弹出“Select Library Files”对话框，开始导入已安装的 Spark 的目录下（例如 spark-2.0.1-bin-hadoop2.6）jars 子目录内的所有 jar 包（通过多选操作），如图 9-16 所示。值得指出的是，Spark-2.0 以下的版本中，这些 jar 包是放在 lib 子目录中的，但是现在改为 jars 子目录中了。

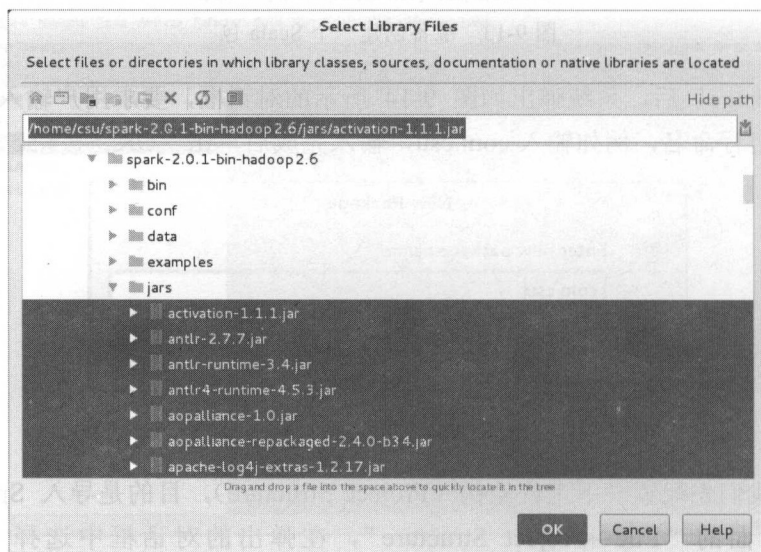


图 9-16 导入 Spark jars 目录下的所有 jar 包

单击图 9-16 的“OK”按钮，并在后续弹出的对话框中一直单击“OK”按钮，回到 IDEA 主界面。这时候，系统会再次进行索引处理。

接着，开始创建 Scala 类。首先将鼠标放在包名（com.csu）上右键单击，弹出列表菜单，然后依次单击“New→Scala class”，在弹出的对话框中，输入类名称，如 sparkPi，并将 Kind 中的内容改为 Object，如图 9-17 所示，单击“OK”按钮。

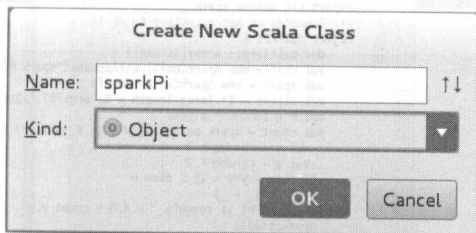


图 9-17 输入类名并将 Kind 设置为 Object

将下面给出的代码复制到如图 9-18 所示的程序编辑区。请注意，IDEA 自动生成了部分代码，如 package com.csu 等，读者复制代码时不要重复。

```
package com.csu
/***** Created by csu on 03/23/17. *****/
import scala.math.random
import org.apache.spark._
/** Computes an approximation to pi */
object sparkPi {
  def main(args: Array[String]) {
    val conf = new SparkConf().setAppName("spark Pi")
    val spark = new SparkContext(conf)
    val slices = if (args.length > 0) args(0).toInt else 2
    val n = 100000 * slices
    val count = spark.parallelize(1 to n, slices).map { i =>
      val x = random * 2 - 1
      val y = random * 2 - 1
      if (x*x + y*y < 1) 1 else 0
    }.reduce(_ + _)
    println("Pi is roughly " + 4.0 * count / n)
    spark.stop()
  }
}
```

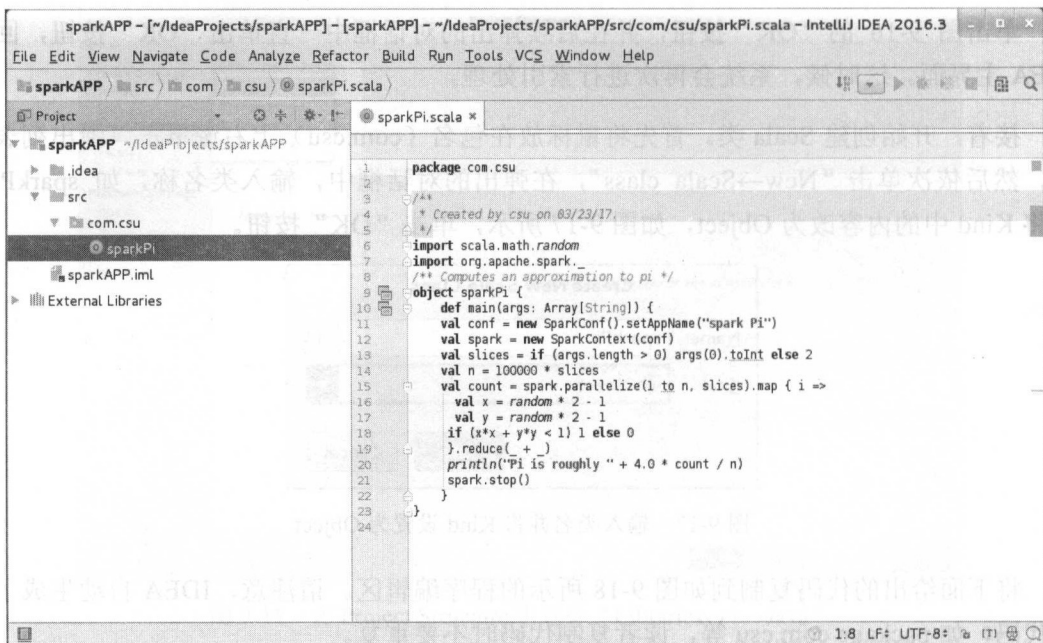



图 9-18 将代码贴入编辑区

如果顺利，编辑区内应当没有错误，这时候可以准备运行程序了。

但是运行之前，还需要修改运行参数。请在主界面的“Run”菜单中选择“Edit Configurations”，如图 9-19 所示。

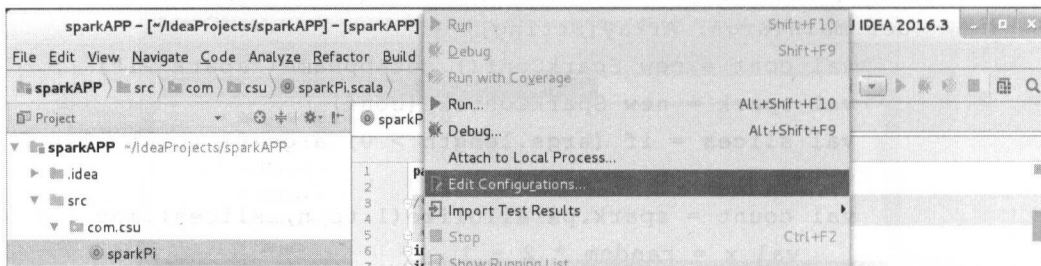


图 9-19 在“Run”菜单中选择“Edit Configurations”准备修改运行参数

单击“Edit Configurations”之后，在弹出的对话框中，单击左上角的绿色“+”，并在弹出的列表中选择“Application”，如图 9-20 所示。

单击选择“Application”后，弹出运行参数设置对话框，如图 9-21 所示。

开始设置参数。其中，Name 设置为 sparkPi；Main class 为 com.csu.sparkPi；VM

options 填写为 “-Dspark.master=local -Dspark.app.name=sparkPi”；Working directory 取系统自动设置的值；Program arguments 不用填写；其余设置保持缺省即可。

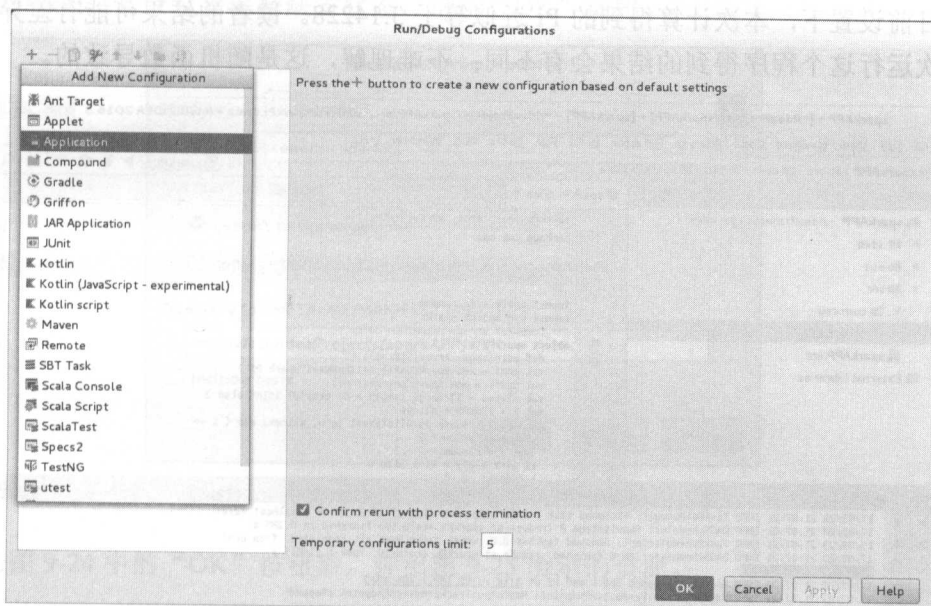


图 9-20 单击左上角的“+”，并在弹出的列表中选择“Application”

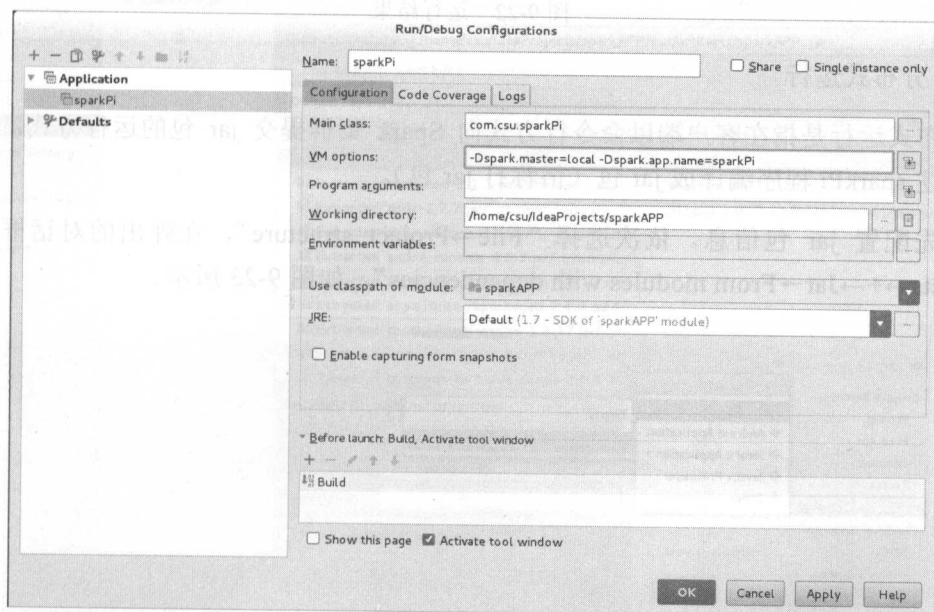


图 9-21 运行参数设置

完成运行参数设置，单击“OK”按钮，回到 IDEA 程序编辑区。右键单击编辑区中的 sparkPi 文件（任意位置），选择“Run sparkPi”即可开始运行程序。图 9-22 是运行结果，在目前设置下，本次计算得到的 Pi 近似等于 3.14228。读者的结果可能有差异。实际上，再次运行这个程序得到的结果会有不同；不难理解，这是随机函数导致的。

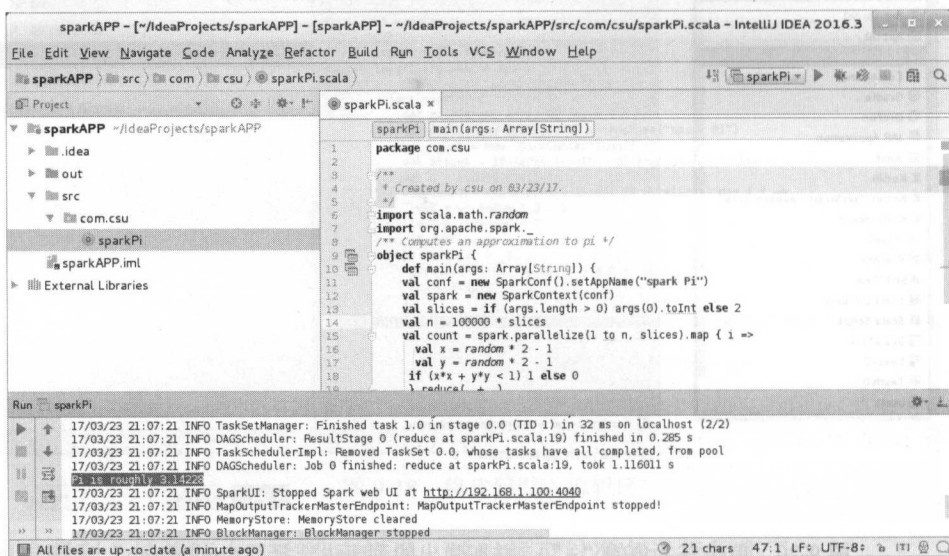


图 9-22 运行结果

3. 分布式运行

分布式运行是指在客户端以命令行方式向 Spark 集群提交 jar 包的运行方式，所以需要将上述 sparkPi 程序编译成 jar 包（俗称打 jar 包）。

首先配置 jar 包信息。依次选择“File→Project structure”，在弹出的对话框中选择“Artifacts→+→Jar→From modules with dependencies”，如图 9-23 所示。

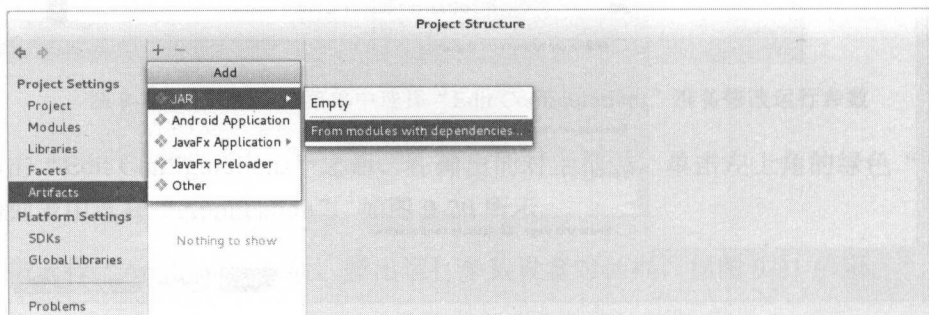


图 9-23 在“Project structure”中选择“Artifacts→+→JAR→From modules with dependencies...”

单击图 9-23 中的“From modules with dependencies”后，弹出图 9-24 所示的“Create JAR from Modules”对话框，可以进行设置。这里我们将 Main Class 设置为 com.csu.sparkPi，其余设置保持缺省状态，单击“OK”按钮。

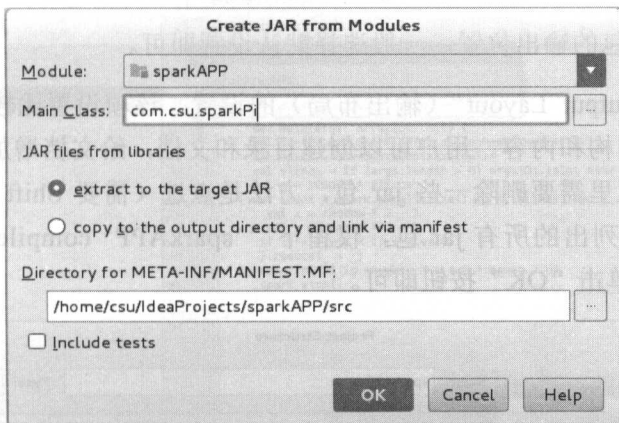


图 9-24 设置主类名、JAR 文件依赖库的输出方式

单击图 9-24 中的“OK”按钮后，弹出图 9-25 所示的页面。

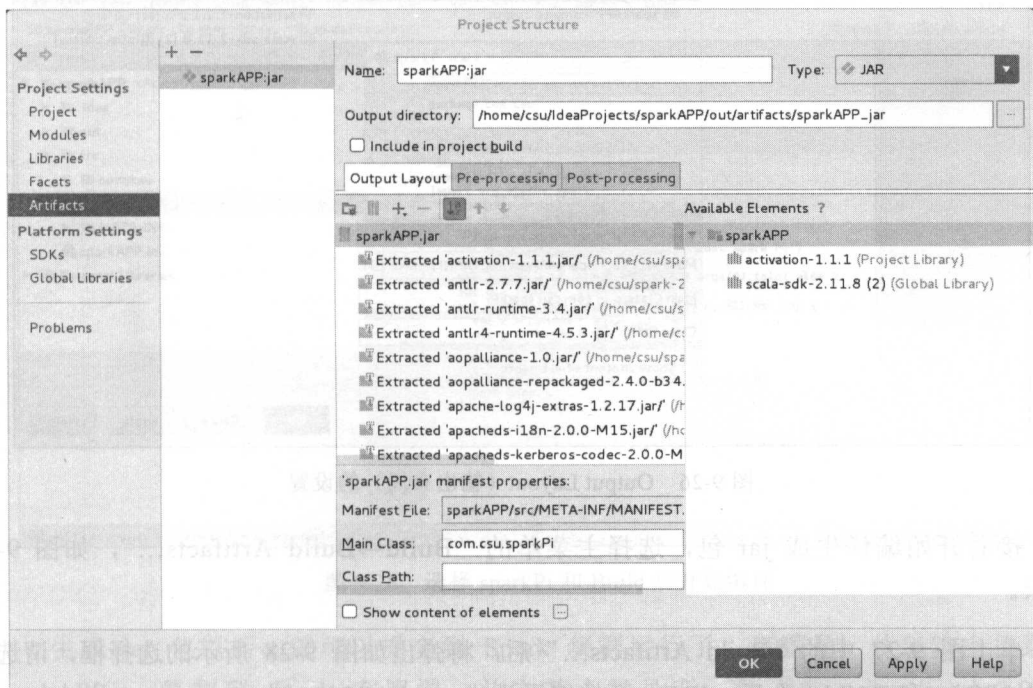


图 9-25 jar 包输出目录设置

在图 9-25 中, 用户需要进行必要的配置。

首先是确定 jar 包的名称 (Name), 可以直接选择默认的 sparkAPP, 也可以修改为其他名称, 例如 MySpark 等, 这里选择 sparkAPP。

其次是确定 jar 包的输出位置, 一般选择默认设置即可。

然后是确定 “Output Layout” (输出布局) 的设置, 该项设置比较关键。输出布局是指生成的 jar 包的结构和内容。用户可以创建目录和文档, 给文档增加或减少模块、库、包、文件等。我们这里需要删除一些 jar 包, 方法是点选 (需要 Shift 或 Ctrl 键的配合) sparkAPP.jar 文档下列出的所有 jar 包, 仅留下 “‘sparkAPP’ compile output” 一项, 如图 9-26 所示, 最后单击 “OK” 按钮即可。

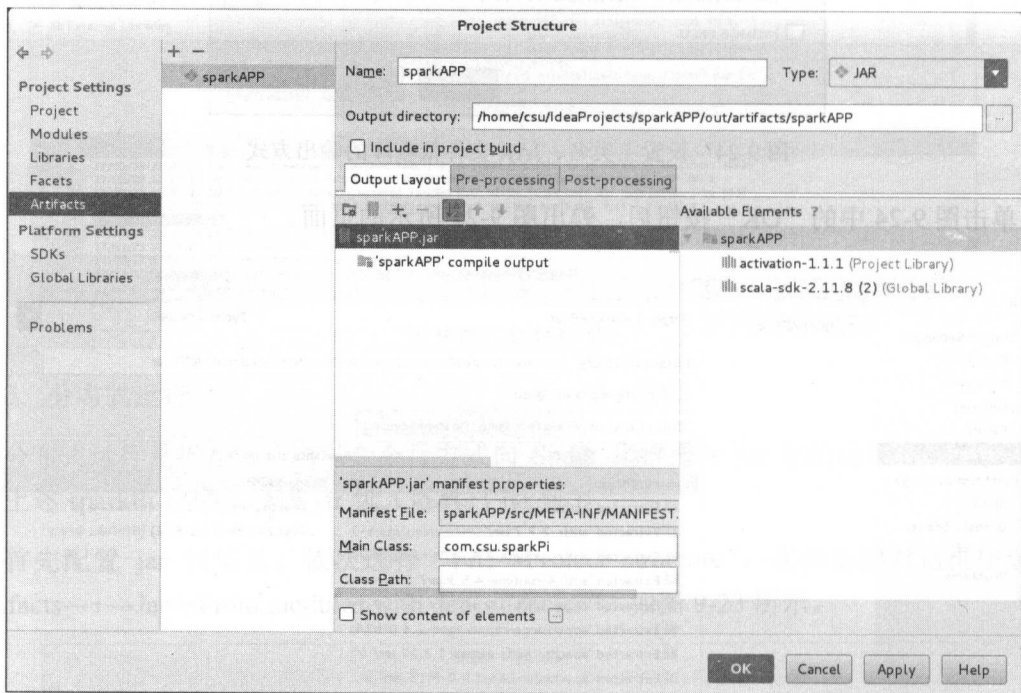


图 9-26 Output Layout (输出布局) 的设置

接着开始编译生成 jar 包, 选择主菜单的 “Build→Build Artifacts...”, 如图 9-27 所示。

单击图 9-27 中的 “Build Artifacts...” 后, 将弹出如图 9-28 所示的选择框, 请选择 sparkAPP.jar 和 Build。之后, IDEA 开始编译生成 sparkAPP.jar 包。

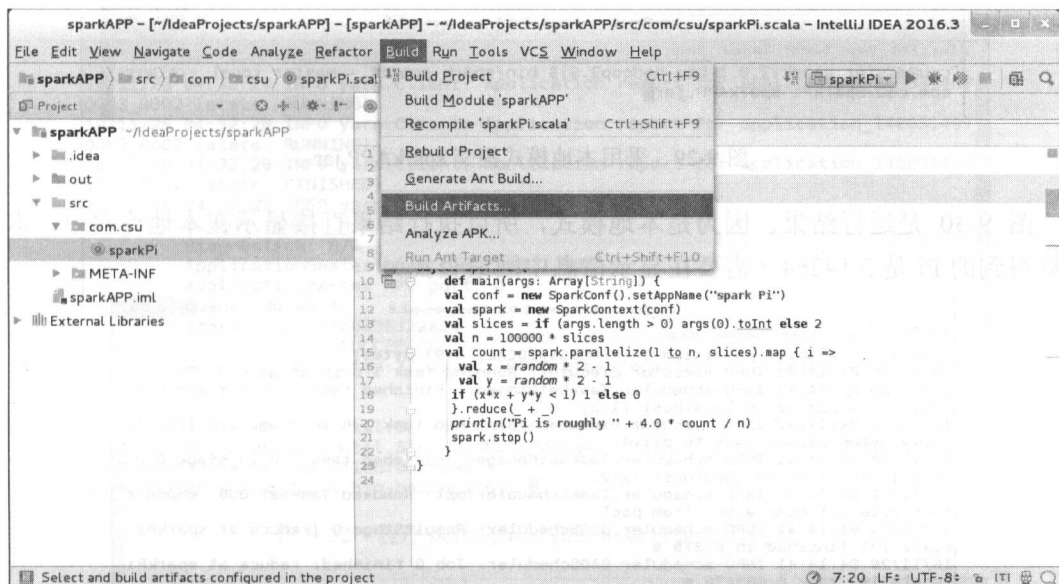


图 9-27 选择主菜单的“Build → Build Artifacts...”

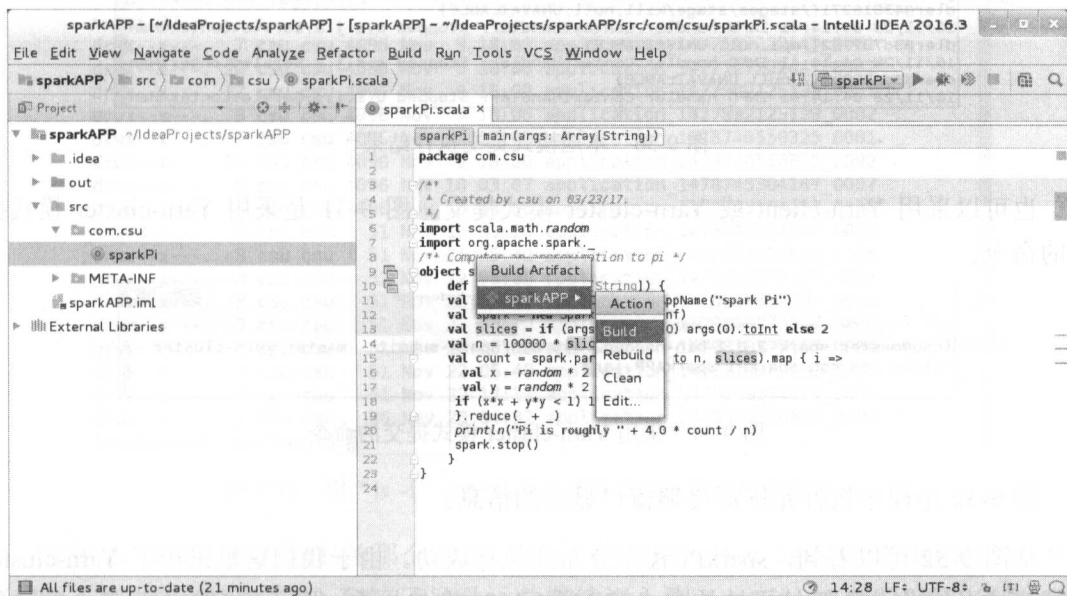


图 9-28 选择 sparkPi 和 Build 后开始编译

生成 jar 包以后，就可以提交到 Spark 集群运行了。步骤是，从包输出目录中将 sparkAPP.jar 复制到 Spark 客服端，放在某个目录下，如“/home/csu/spark-2.0.1-bin-hadoop2.6/”。接着进入 Spark 安装目录，执行如图 9-29 所示的命令。

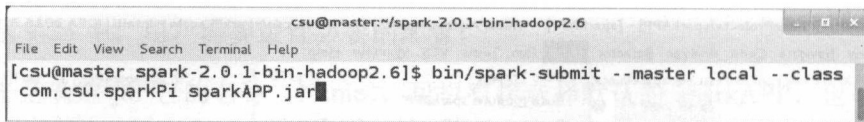


图 9-29 采用本地模式提交 sparkAPP.jar

图 9-30 是运行结果，因为是本地模式，所以执行结果直接显示在本地终端中，本次计算得到的 Pi 是 3.14264（需要在显示信息中回滚查找）。

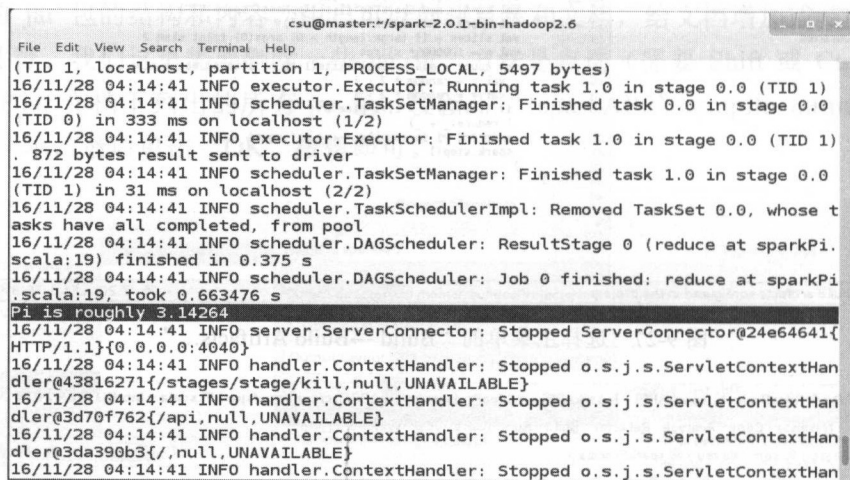


图 9-30 本地模式提交执行结果

也可以采用 Yarn-client 或 Yarn-cluster 模式提交。图 9-31 是采用 Yarn-cluster 模式提交的命令。

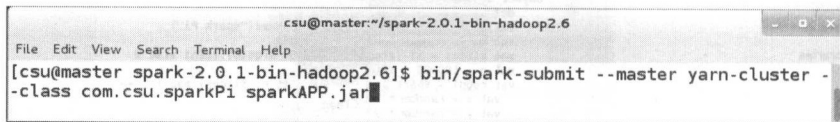
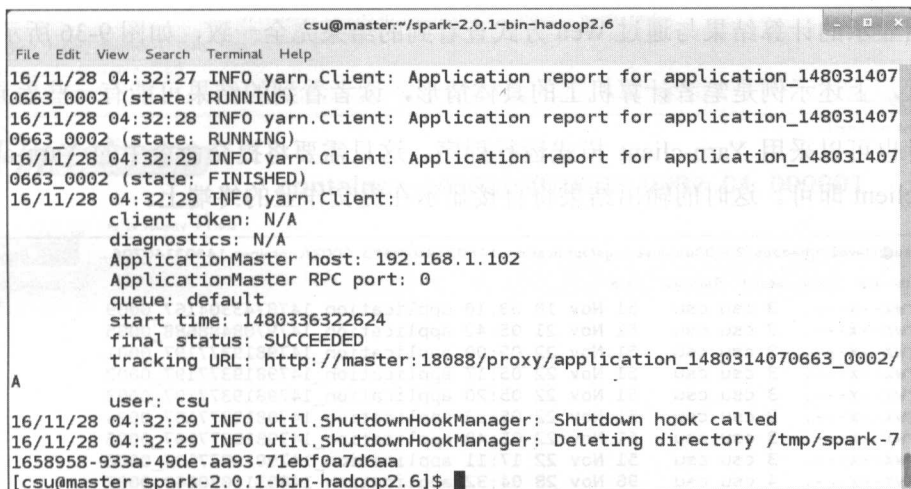


图 9-31 采用 Yarn-cluster 模式提交的命令

图 9-32 是程序执行完毕后终端窗口显示的信息。

从图 9-32 可以看到，sparkPi 程序分布式执行成功。由于我们这里采用了 Yarn-cluster 运行模式，因此，计算结果被放在 192.168.1.102 节点上了。除了我们前面已经介绍过的通过 URL（如图 9-32 中的 http://master:18088/proxy/application_1480314070663_0002/A）查看执行结果以外，读者还可以 SSH 连接到 Slave，或直接到该节点上开一个终端窗口，然后进入其 Hadoop 安装目录下的“/logs/userlogs”子目录，用“ls -l”命令显示该节点上的应用程序日志目录，来查看执行结果，如图 9-33 所示。

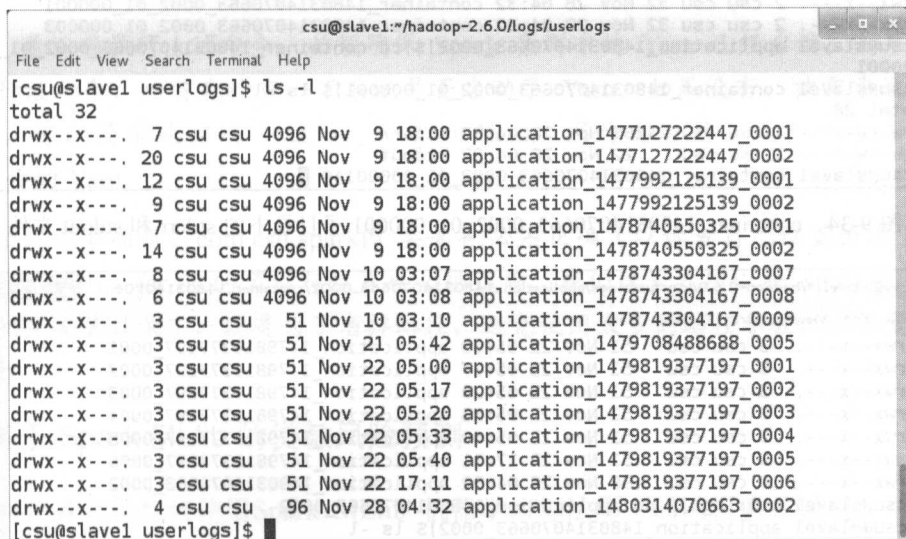


```

csu@master:~/spark-2.0.1-bin-hadoop2.6
File Edit View Search Terminal Help
16/11/28 04:32:27 INFO yarn.Client: Application report for application_148031407
0663_0002 (state: RUNNING)
16/11/28 04:32:28 INFO yarn.Client: Application report for application_148031407
0663_0002 (state: RUNNING)
16/11/28 04:32:29 INFO yarn.Client: Application report for application_148031407
0663_0002 (state: FINISHED)
16/11/28 04:32:29 INFO yarn.Client:
  client token: N/A
  diagnostics: N/A
  ApplicationMaster host: 192.168.1.102
  ApplicationMaster RPC port: 0
  queue: default
  start time: 1480336322624
  final status: SUCCEEDED
  tracking URL: http://master:18088/proxy/application_1480314070663_0002/
A
  user: csu
16/11/28 04:32:29 INFO util.ShutdownHookManager: Shutdown hook called
16/11/28 04:32:29 INFO util.ShutdownHookManager: Deleting directory /tmp/spark-7
1658958-933a-49de-aa93-71ebf0a7d6aa
[csu@master spark-2.0.1-bin-hadoop2.6]$

```

图 9-32 程序执行完毕后在终端窗口显示的信息



```

csu@slave1:~/hadoop-2.6.0/logs/userlogs
File Edit View Search Terminal Help
[csu@slave1 userlogs]$ ls -l
total 32
drwx--x---, 7 csu csu 4096 Nov 9 18:00 application_1477127222447_0001
drwx--x---, 20 csu csu 4096 Nov 9 18:00 application_1477127222447_0002
drwx--x---, 12 csu csu 4096 Nov 9 18:00 application_1477992125139_0001
drwx--x---, 9 csu csu 4096 Nov 9 18:00 application_1477992125139_0002
drwx--x---, 7 csu csu 4096 Nov 9 18:00 application_1478740550325_0001
drwx--x---, 14 csu csu 4096 Nov 9 18:00 application_1478740550325_0002
drwx--x---, 8 csu csu 4096 Nov 10 03:07 application_1478743304167_0007
drwx--x---, 6 csu csu 4096 Nov 10 03:08 application_1478743304167_0008
drwx--x---, 3 csu csu 51 Nov 10 03:10 application_1478743304167_0009
drwx--x---, 3 csu csu 51 Nov 21 05:42 application_1479708488688_0005
drwx--x---, 3 csu csu 51 Nov 22 05:00 application_1479819377197_0001
drwx--x---, 3 csu csu 51 Nov 22 05:17 application_1479819377197_0002
drwx--x---, 3 csu csu 51 Nov 22 05:20 application_1479819377197_0003
drwx--x---, 3 csu csu 51 Nov 22 05:33 application_1479819377197_0004
drwx--x---, 3 csu csu 51 Nov 22 05:40 application_1479819377197_0005
drwx--x---, 3 csu csu 51 Nov 22 17:11 application_1479819377197_0006
drwx--x---, 4 csu csu 96 Nov 28 04:32 application_1480314070663_0002
[csu@slave1 userlogs]$

```

图 9-33 用“ls -l”命令显示计算节点上应用程序日志目录

在图 9-33 中，前面提到的那个 URL 中包含的 application_1480314070663_0002 作为子目录正好位于最后。进入到该子目录，并继续深入到 container_1480314070663_0002_01_000001 子目录，就可以最终看到 stderr 和 stdout 两个文件，如图 9-34 所示。

最后，利用 cat 命令显示 stdout 文件内容，即可看到计算结果，显示内容是“Pi is roughly 3.13902”，如图 9-35 所示。

上图显示的计算结果与通过 Web 方式查看到的结果完全一致, 如图 9-36 所示。

注意, 上述示例是笔者计算机上的具体情形, 读者看到的结果可能有一些差异。

读者也可以采用 Yarn-client 模式运行程序, 这只需要将提交命令中的 Yarn-cluster 改为 Yarn-client 即可。这时的输出结果将直接显示在提交作业的终端上。

```
csu@slave1:~/hadoop-2.6.0/logs/userlogs/application_1480314070663_0002/container_14803140706...
File Edit View Search Terminal Help
drwx--x---. 3 csu csu 51 Nov 10 03:10 application_1478743304167_0009
drwx--x---. 3 csu csu 51 Nov 21 05:42 application_1479708488688_0005
drwx--x---. 3 csu csu 51 Nov 22 05:00 application_1479819377197_0001
drwx--x---. 3 csu csu 51 Nov 22 05:17 application_1479819377197_0002
drwx--x---. 3 csu csu 51 Nov 22 05:20 application_1479819377197_0003
drwx--x---. 3 csu csu 51 Nov 22 05:33 application_1479819377197_0004
drwx--x---. 3 csu csu 51 Nov 22 05:40 application_1479819377197_0005
drwx--x---. 3 csu csu 51 Nov 22 17:11 application_1479819377197_0006
drwx--x---. 4 csu csu 96 Nov 28 04:32 application_1480314070663_0002
[csu@slave1 userlogs]$ cd application_1480314070663_0002
[csu@slave1 application_1480314070663_0002]$ ls -l
total 0
drwx--x---. 2 csu csu 32 Nov 28 04:32 container_1480314070663_0002_01_000001
drwx--x---. 2 csu csu 32 Nov 28 04:32 container_1480314070663_0002_01_000003
[csu@slave1 application_1480314070663_0002]$ cd container_1480314070663_0002_01_000001
[csu@slave1 container_1480314070663_0002_01_000001]$ ls -l
total 28
-rw-rw-r--. 1 csu csu 23265 Nov 28 04:32 stderr
-rw-rw-r--. 1 csu csu 22 Nov 28 04:32 stdout
[csu@slave1 container_1480314070663_0002_01_000001]$
```

图 9-34 container_1480314070663_0002_01_000001 子目录下的 stderr 和 stdout 文件

```
csu@slave1:~/hadoop-2.6.0/logs/userlogs/application_1480314070663_0002/container_14803140706...
File Edit View Search Terminal Help
drwx--x---. 3 csu csu 51 Nov 22 05:00 application_1479819377197_0001
drwx--x---. 3 csu csu 51 Nov 22 05:17 application_1479819377197_0002
drwx--x---. 3 csu csu 51 Nov 22 05:20 application_1479819377197_0003
drwx--x---. 3 csu csu 51 Nov 22 05:33 application_1479819377197_0004
drwx--x---. 3 csu csu 51 Nov 22 05:40 application_1479819377197_0005
drwx--x---. 3 csu csu 51 Nov 22 17:11 application_1479819377197_0006
drwx--x---. 4 csu csu 96 Nov 28 04:32 application_1480314070663_0002
[csu@slave1 userlogs]$ cd application_1480314070663_0002
[csu@slave1 application_1480314070663_0002]$ ls -l
total 0
drwx--x---. 2 csu csu 32 Nov 28 04:32 container_1480314070663_0002_01_000001
drwx--x---. 2 csu csu 32 Nov 28 04:32 container_1480314070663_0002_01_000003
[csu@slave1 application_1480314070663_0002]$ cd container_1480314070663_0002_01_000001
[csu@slave1 container_1480314070663_0002_01_000001]$ ls -l
total 28
-rw-rw-r--. 1 csu csu 23265 Nov 28 04:32 stderr
-rw-rw-r--. 1 csu csu 22 Nov 28 04:32 stdout
[csu@slave1 container_1480314070663_0002_01_000001]$ cat stdout
Pi is roughly 3.13902
[csu@slave1 container_1480314070663_0002_01_000001]$
```

图 9-35 用 cat 命令显示 stdout 文件内容

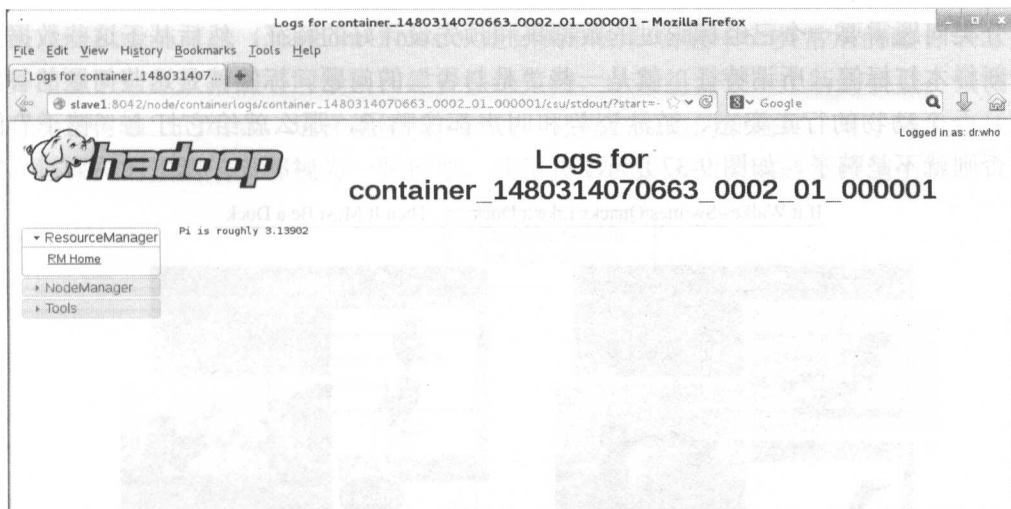


图 9-36 通过 Web 查看到的执行结果

注意：要正确执行上述命令，必须确保 Spark 集群中的 Yarn 已经启动。如果读者不能成功执行该命令，应首先检查一下系统是否启动了 Yarn。如果没有，就需要设法启动它。例如，可以简单地启动 Hadoop 来启动 Yarn，因为有时候，用户可能只安装了 Hadoop 自带的 Yarn。

另一个注意事项是，`com.csu.sparkPi` 类名也必须正确（以读者自己设置为准），而且区分大小写；同时，`jar` 包的名称与位置也必须正确，我们这里之所以将 `sparkAPP.jar` 放在 Spark 的主安装目录下，就是为了省掉路径，避免使用太多的路径参数，以尽量降低出错的可能性。

9.2.2 基于 Spark MLlib 的贷款风险预测

本节向读者介绍如何使用 Spark MLlib 机器学习算法库中的随机森林模型，对银行信用贷款的风险做分类预测。该示例有很好的应用参考价值。

Spark MLlib 机器学习算法库基于 `DataFrame`。`DataFrame` 提供了大量接口，帮助用户创建和调优机器学习流程。用户通过 `DataFrame` 使用 MLlib，能够实现模型的智能优化，从而提升模型效果。

1. 特征与标签

分类算法是一类有监督的机器学习算法，它根据已知标签的样本（例如，已经明确交易是否存在欺诈）来预测其他样本所属的类别（是否属于欺诈性的交易）。

分类问题需要一个已经标记过的数据集和预先设计好的特征，然后基于这些数据来学习给新样本打标签。所谓特征，就是一些“是与否”的问题。标签就是这些问题的答案。例如，一个动物的行走姿态、游泳姿势和叫声都像鸭子，那么就给它打上“鸭子”的标签，否则就不是鸭子，如图 9-37 所示。

If it Walkes/Swimes/Quacks Like a Duck..... Then It Must Be a Duck

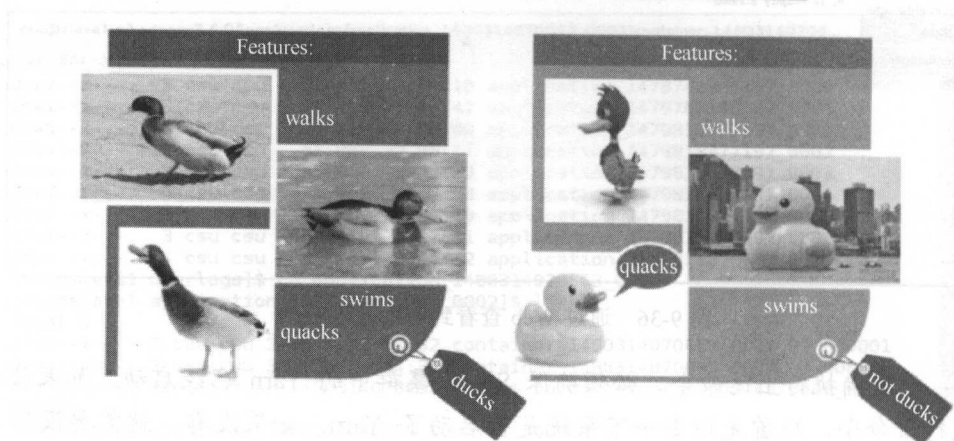


图 9-37 鸭子的特征与标签

来看一个银行信贷风险预测的例子。我们需要预测某个人是否会按时还款，这就是标签，即此人的信用度。用来预测“是与否”的属性就是申请人的基本信息和社会身份信息，包括职业、年龄、储蓄存款、婚姻状态等，这些就是特征。特征可以用来构建一个分类模型，开发人员从中提取出对分类有帮助的信息。

2. 决策树模型

决策树是一种基于输入特征来预测类别或者标签的分类模型，其工作原理是这样的，它在每个节点都需要计算特征在该节点的表达式值，然后基于运算结果选择一个分支通往下一个节点。图 9-38 展示了一种用来预测信用风险的决策树模型。

每个决策问题就是模型的一个节点，“是”或者“否”的答案是通往子节点的分支。在图 9-32 给出的决策树中，第一个问题是：账户余额是否大于 200 元？如果答案是否定的，那么就需要继续解答第二个问题，即当前就职时间是否超过 1 年？如果答案也是否定的，那么该用户是不可信赖的，也就是该用户的信用标签是否定的。

3. 随机森林模型

在机器学习算法中，融合学习算法结合了多个机器学习的算法，能够得到更好的分类

效果。随机森林算法（Random Forest）是分类和回归问题中一类常用的融合学习方法，该算法基于训练数据的不同子集构建多棵决策树，从而组合成一个新的模型。预测结果是所有决策树输出的组合，这样就能减少波动，并提高预测的准确度。对于随机森林分类模型，每棵树的预测结果都视为一张投票，获得投票数最多的类别就是预测的类别。

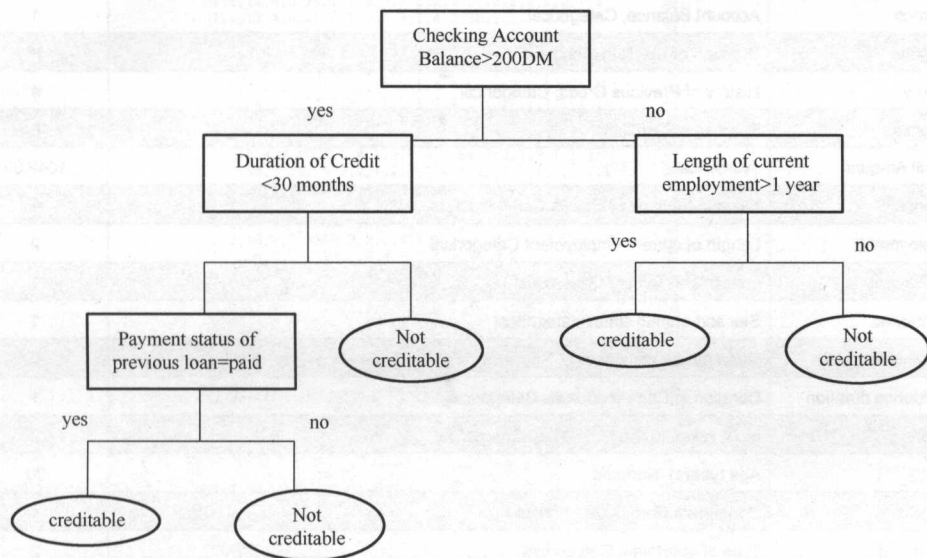


图 9-38 一个用来预测信用风险的决策树模型

图 9-39 是随机森林模型的示意。

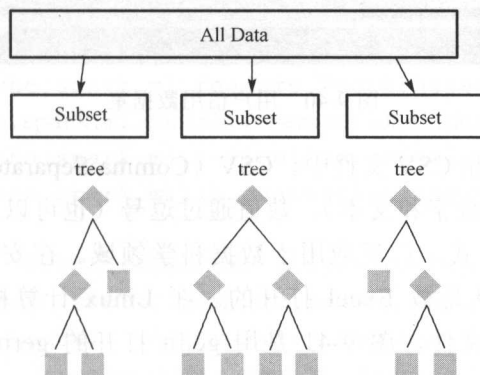


图 9-39 随机森林模型

4. 数据集

本节使用经典的德国人信用度数据集。该数据集按照一系列特征属性将用户的信用风

险标签分为好和坏两类。图 9-40 给出了用户信用数据集的特征，共有 1 个标签和 20 个特征，其中 `creditability` 是信用标签，取值为 0 或 1，代表不可信和可信两个类别。

Feature	Description	Example
Creditability	label	0,1
Balance	Account Balance, Categorical	1
Duration	Duration of Credit in Months	18
History	History of Previous Credit, Categorical	4
Purpose	Purpose of Credit	0
Credit Amount	Numerical	1049.00
Savings	Value in Savings or Stocks Categorical	1
Employment	Length of current employment Categorical	2
instPercent	Installment Percent Categorical	1
sexMarried	Sex and Marital status categorical	1
guarantors	Guarantors categorical	0
Residence duration	Duration in Current address Categorical	1
assets	Must valuable available asset Categorical	2
age	Age (years) Numeric	20
concCredit	Concurrent Credits Categorical	1
apartment	Type of apartment Categorical	2
credits	No of Credits at this Bank, Numeric	1
occupation	categorical	1
dependents	No of dependents, Numeric	2
has Phone	Has phone categorical	1
foreign	Foreign worker	1

图 9-40 用户信用数据集

用户信用数据集存放在 CSV 文件中。CSV (Comma-Separated Values) 是一种文本文件，用于存储表格数据（数字和文本），数值通过逗号（也可以用其他符号）分隔。CSV 是一种通用的简单文件格式，广泛应用于数据科学领域。在安装了 Microsoft Excel 的计算机上，CSV 文件默认是被 Excel 打开的。在 Linux 计算机上，用户可以用任何文本编辑器打开一个 CSV 文件，图 9-41 是用 gedit 打开的 `germancredit.csv` 文件，共有 1000 条记录。

5. 运行环境与程序

用于处理上述 `germancredit.csv` 文件的基于 Spark MLlib 的 Scala 语言应用程序如下。

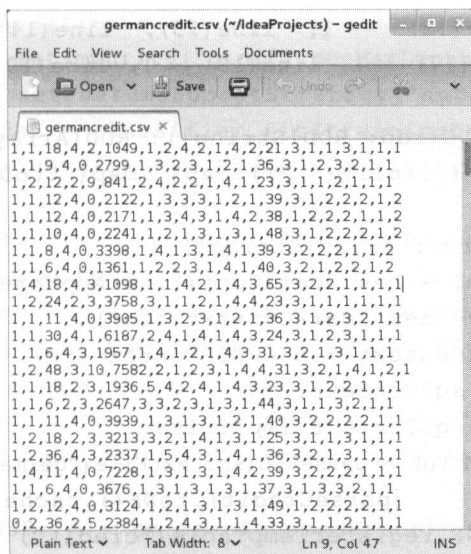


图 9-41 用 gedit 打开的 germancredit.csv 文件

```
import org.apache.spark._
import org.apache.spark.rdd.RDD
import org.apache.spark.sql.SQLContext
import org.apache.spark.sql.functions._
import org.apache.spark.sql.types._
import org.apache.spark.sql._
import org.apache.spark.ml.classification.RandomForestClassifier
import org.apache.spark.ml.evaluation.BinaryClassificationEvaluator
import org.apache.spark.ml.feature.StringIndexer
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.tuning.{ ParamGridBuilder, CrossValidator }
import org.apache.spark.ml.{ Pipeline, PipelineStage }
import org.apache.spark.mllib.evaluation.ReggressionMetrics

object Credit {
  case class Credit(creditability: Double, balance: Double,
    duration: Double, history: Double, purpose: Double, amount: Double, savings:
    Double, employment: Double, instPercent: Double, sexMarried: Double,
    guarantors: Double, residenceDuration: Double, assets: Double, age: Double,
    concCredit: Double, apartment: Double, credits: Double, occupation: Double,
    dependents: Double, hasPhone: Double, foreign: Double)

  def parseCredit(line: Array[Double]): Credit = {
    Credit( line(0), line(1) - 1, line(2), line(3), line(4),
    line(5), line(6) - 1, line(7) - 1, line(8), line(9) - 1, line(10) - 1,
```

```

line(11) - 1, line(12) - 1, line(13), line(14) - 1, line(15) -
1, line(16) - 1, line(17) - 1, line(18) - 1, line(19) - 1, line(20) - 1 )
    }
    def parserRDD(rdd: RDD[String]): RDD[Array[Double]] = {
        rdd.map(_ .split(",")).map(_ .map(_ .toDouble))
    }
    def main(args: Array[String]) {
        val conf = new SparkConf().setAppName("SparkDFebay")
        val sc = new SparkContext(conf)
        val sqlContext = new SQLContext(sc)
        import sqlContext._
        import sqlContext.implicits._
        val creditDF = parserRDD(sc.textFile("germancredit.csv")).map
            (parseCredit).toDF().cache()
        creditDF.registerTempTable("credit")
        creditDF.printSchema
        creditDF.show
        sqlContext.sql("SELECT creditability, avg(balance) as avgbalance,
avg(amount) as avgamt, avg(duration) as avgdur FROM credit GROUP BY
creditability ").show
        creditDF.describe("balance").show
        creditDF.groupBy("creditability").avg("balance").show
        val featureCols = Array("balance", "duration", "history",
            "purpose", "amount", "savings", "employment",
            "instPercent", "sexMarried", "guarantors",
            "residenceDuration", "assets", "age",
            "concCredit", "apartment", "credits",
            "occupation", "dependents", "hasPhone",
            "foreign")
        val assembler = new VectorAssembler().setInputCols(featureCols).
            setOutputCol("features")
        val df2 = assembler.transform(creditDF)
        df2.show
        val labelIndexer = new StringIndexer().
            setInputCol("creditability").setOutputCol("label")
        val df3 = labelIndexer.fit(df2).transform(df2)
        df3.show
        val splitSeed = 5043
        val Array(trainingData, testData) = df3.randomSplit(Array(0.7,
            0.3), splitSeed)
    }
}

```

```

val classifier = new RandomForestClassifier().
    setImpurity("gini").setMaxDepth(3).setNumTrees(20).
    setFeatureSubsetStrategy("auto").setSeed(5043)
val model = classifier.fit(trainingData)
val evaluator = new BinaryClassificationEvaluator().
    setLabelCol("label")

val predictions = model.transform(testData)
model.toDebugString
val accuracy = evaluator.evaluate(predictions)
println("accuracy before pipeline fitting" + accuracy)
val rm = new RegressionMetrics(
    predictions.select("prediction", "label").rdd.map(x =>
        (x(0).asInstanceOf[Double], x(1).asInstanceOf[Double]))
)
println("MSE: " + rm.meanSquaredError)
println("MAE: " + rm.meanAbsoluteError)
println("RMSE Squared: " + rm.rootMeanSquaredError)
println("R Squared: " + rm.r2)
println("Explained Variance: " + rm.explainedVariance + "\n")
val paramGrid = new ParamGridBuilder()
    .addGrid(classifier.maxBins, Array(25, 31))
    .addGrid(classifier.maxDepth, Array(5, 10))
    .addGrid(classifier.numTrees, Array(20, 60))
    .addGrid(classifier.impurity, Array("entropy", "gini"))
    .build()
val steps: Array[PipelineStage] = Array(classifier)
val pipeline = new Pipeline().setStages(steps)
val cv = new CrossValidator()
    .setEstimator(pipeline)
    .setEvaluator(evaluator)
    .setEstimatorParamMaps(paramGrid)
    .setNumFolds(20)
val pipelineFittedModel = cv.fit(trainingData)
val predictions2 = pipelineFittedModel.transform(testData)
val accuracy2 = evaluator.evaluate(predictions2)
println("accuracy after pipeline fitting" + accuracy2)
println(pipelineFittedModel.bestModel.asInstanceOf[
    org.apache.spark.ml.PipelineModel].stages(0))
pipelineFittedModel
    .bestModel.asInstanceOf[org.apache.spark.ml.PipelineModel]

```

```

    .stages(0)
    .extractParamMap
    val rm2 = new RegressionMetrics(
        predictions2.select("prediction", "label").rdd.map(x =>
            (x(0).asInstanceOf[Double], x(1).asInstanceOf[Double]))
    )
    println("MSE: " + rm2.meanSquaredError)
    println("MAE: " + rm2.meanAbsoluteError)
    println("RMSE Squared: " + rm2.rootMeanSquaredError)
    println("R Squared: " + rm2.r2)
    println("Explained Variance: " + rm2.explainedVariance + "\n")
}
}

```

读者可以从链接 <https://github.com/caroljmcDonald/spark-ml-randomforest-creditrisk> 中下载上述代码和数据文件（germancredit.csv）。

下面利用上述程序和数据，构建一个由决策树组成的随机森林模型，用以预测用户的信用标签类别。

本书使用 Spark 2.0.1，该版本以上的 Spark 也没有问题。读者需要在 Master 节点的用户目录下创建一个工作目录（例如采用 IEDA 自动创建的“/home/csu/IdeaProjects”），然后将 germancredit.csv 文件复制到该目录下。接着启动 IEDA 集成开发环境，单击“Create New Project”，创建一个 Scala 新工程，用户可自行命名，如 Credit，如图 9-42 所示。

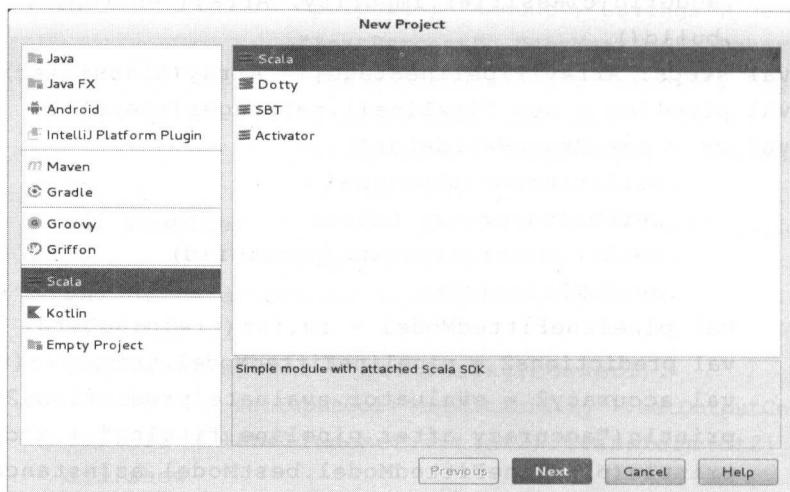


图 9-42 创建一个 Scala 新工程（选择完毕后单击 next）

在图 9-42 中，选择 Scala 后单击“Next”按钮进入工程基本设置页面，如图 9-43 所示。

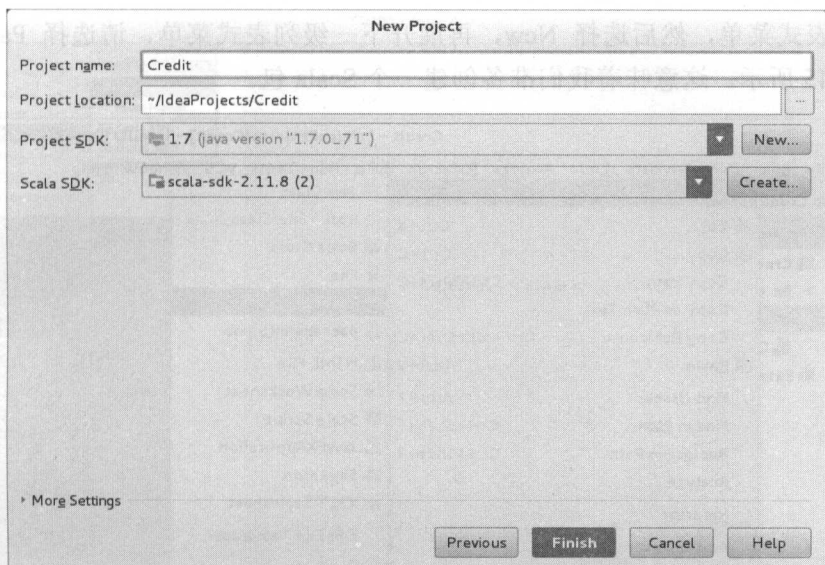


图 9-43 工程名称与位置等基本设置

在图 9-43 中，开发人员需要设置 Project name、Project location 和 Project SDK 等。这里我们给工程取名为 Credit（读者可以任意取一个名称），工程位置（Project location）自动设置。而 Project SDK 与 Scala SDK 是前面安装 IDEA 时已经设置好的，所以会自动显示出来。单击“Finish”按钮后，返回如图 9-44 所示的主界面。

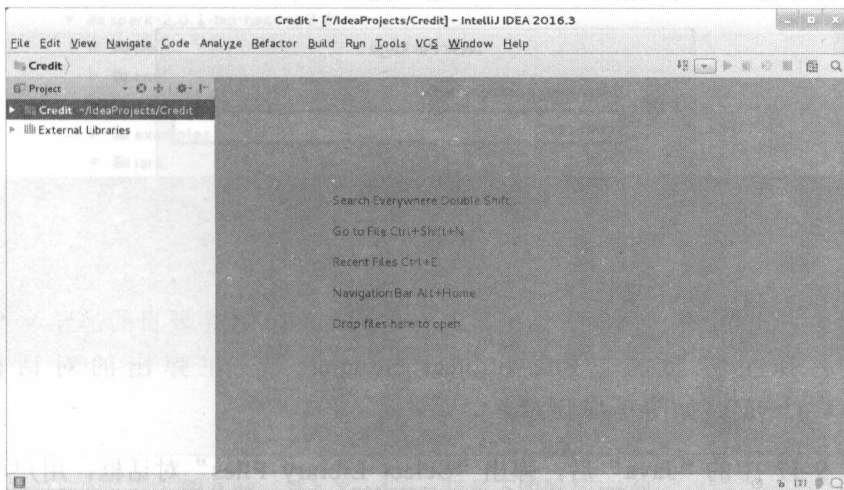


图 9-44 进入开发环境主界面

在图 9-44 中，在左边窗口中展开 Credit 工程，然后将鼠标放在 src 上，（右键单击鼠

标) 展开列表式菜单, 然后选择 **New**, 再展开下一级列表式菜单, 请选择 **Package** 并单击, 如图 9-45 所示。这意味着我们准备创建一个 Scala 包。

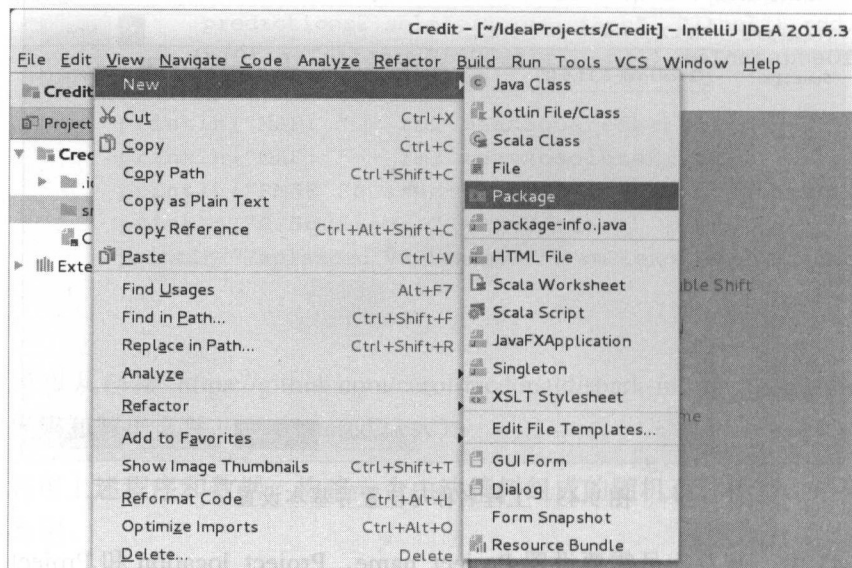


图 9-45 创建一个 Scala 包

选择 **Package** 后, 系统弹出如图 9-46 所示的对话框, 要求用户输入新的 **Package** 名称, 用户可自行输入, 例如输入 **com.csu**, 输入完成后单击“OK”按钮。

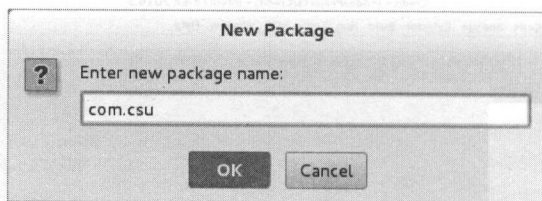


图 9-46 输入新的 Package 名称

接下来, 我们先配置一下工程结构 (Project Structure), 主要目的是导入 **Spark** 依赖包。依次选择主界面的“**File→Project Structure**”, 在弹出的对话框中选择“**Libraries→++→Java**”, 如图 9-47 所示。

单击图 9-47 中的“**Java**”后, 弹出“**Select Library Files**”对话框, 用户可以开始导入安装好的 **Spark** 目录 (例如 **spark-2.0.1-bin-hadoop2.6**) 下 **jars** 内的所有 **jar** 包, 如图 9-48 所示。

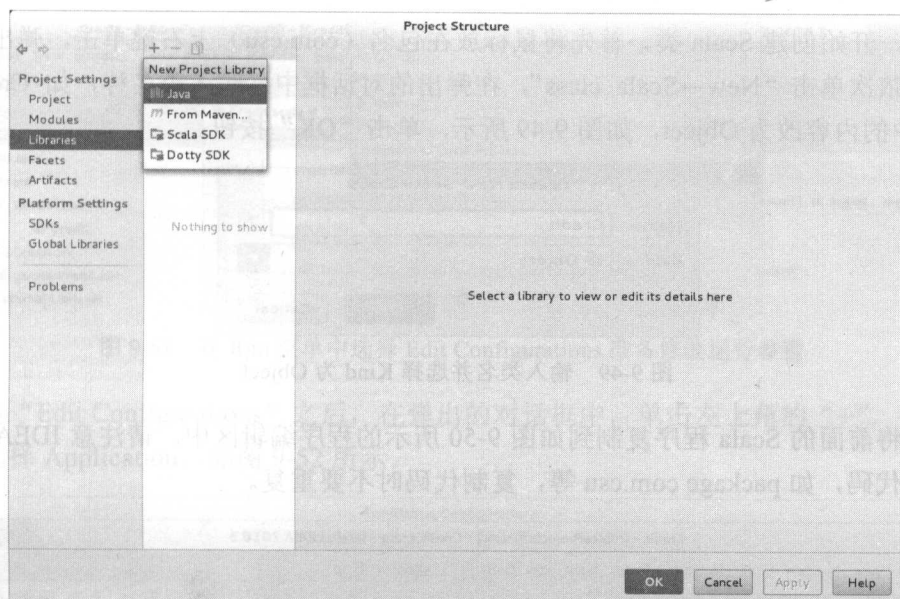


图 9-47 在 Project Structure 中选择“Libraries→+→Java”

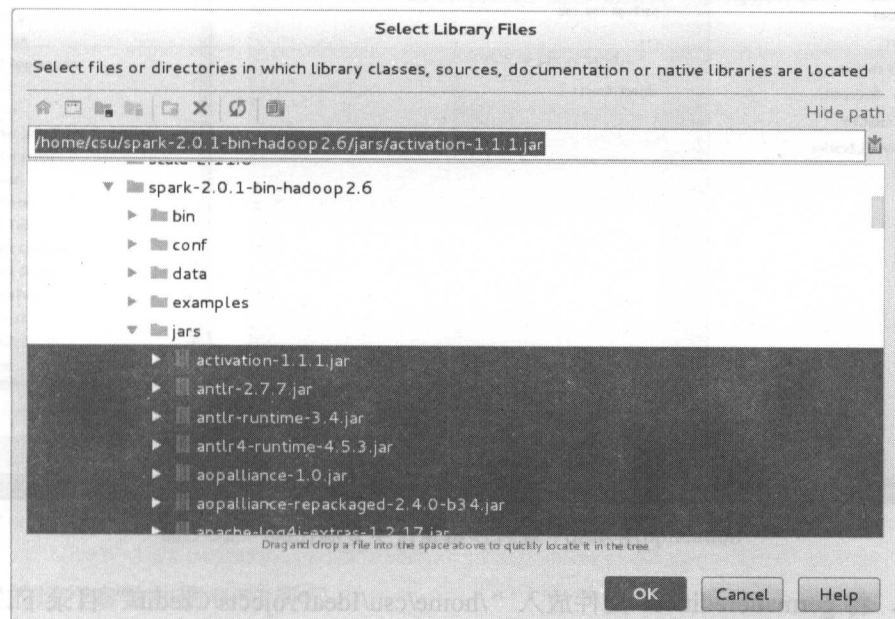


图 9-48 导入 Spark jars 目录下的所有 jar 包

单击图 9-48 中的“OK”按钮，并在后续弹出的对话框中一直单击“OK”按钮，回到 IDEA 主界面即可。

接着,开始创建 Scala 类。首先将鼠标放在包名 (com.csu) 上右键单击,弹出列表菜单,然后依次单击“New→Scala class”,在弹出的对话框中,输入类名称,如 Credit,并将 Kind 中的内容改为 Object,如图 9-49 所示,单击“OK”按钮。

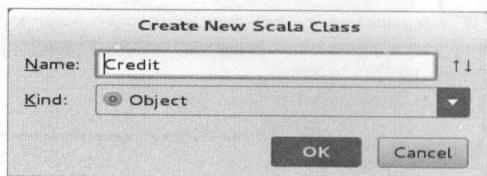


图 9-49 输入类名并选择 Kind 为 Object

然后将前面的 Scala 程序复制到如图 9-50 所示的程序编辑区中。请注意 IDEA 自动生成的部分代码,如 package com.csu 等,复制代码时不要重复。

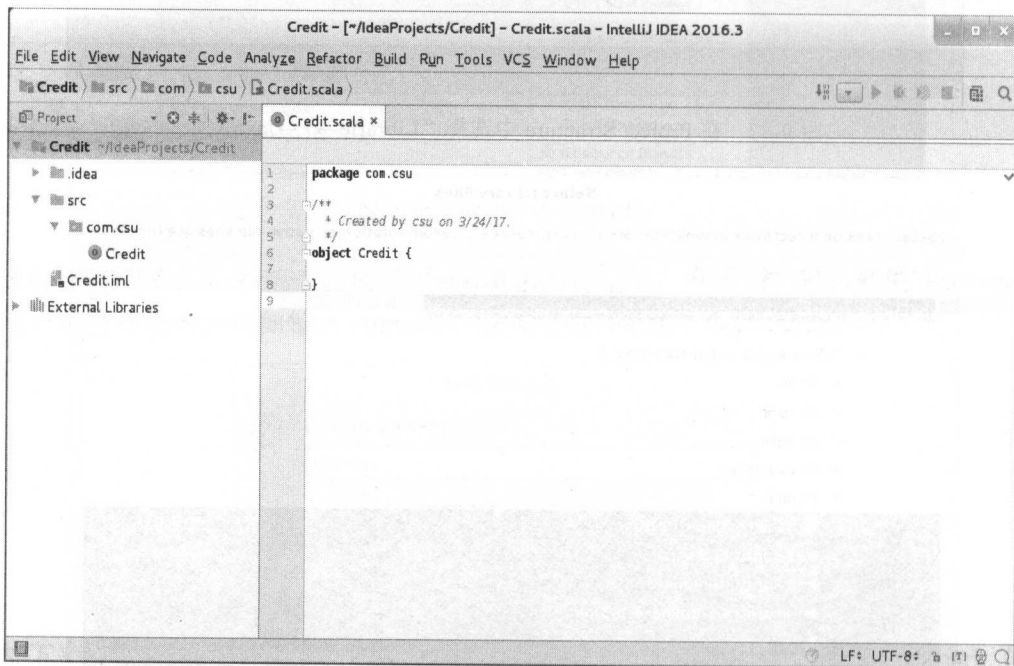


图 9-50 在程序编辑区中复制代码

接着,将 germancredit.csv 文件放入“/home/csu/IdeaProjects/Credit/”目录下。

如果顺利,编辑区内应当没有错误,这时候可以准备运行程序了。

但是运行之前,还需要修改运行参数设置。请在主界面的 Run 菜单中选择 Edit Configurations,如图 9-51 所示。

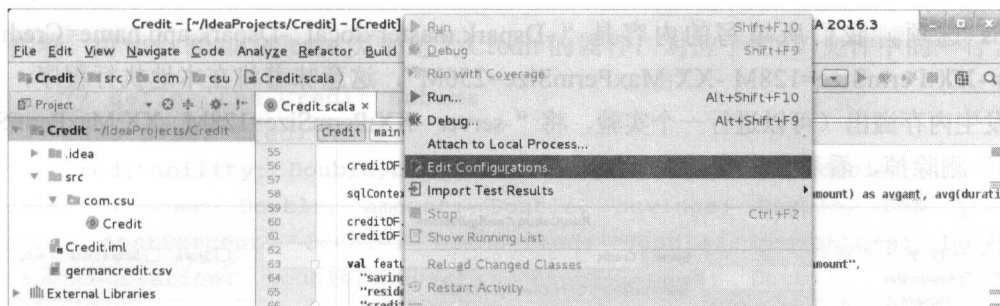


图 9-51 在 Run 菜单中选择 Edit Configurations 准备修改运行参数

单击“Edit Configurations”之后，在弹出的对话框中，单击左上角的“+”，并在左边列表中选择 Application，如图 9-52 所示。

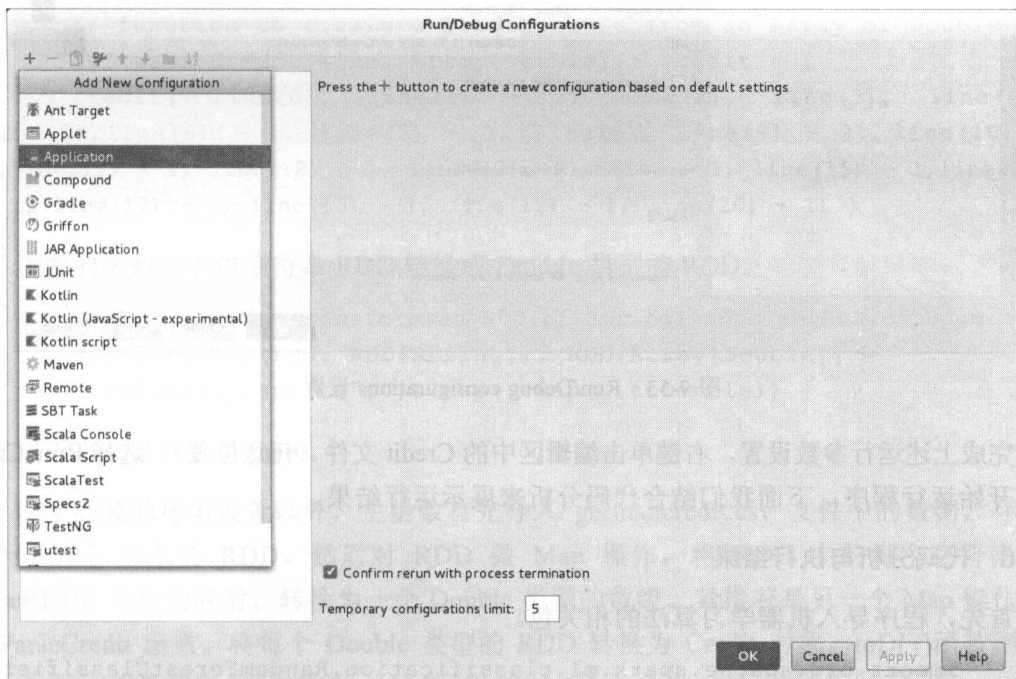


图 9-52 单击“+”，并在左边列表中选择 Application

设置的运行参数如图 9-53 所示。

其中，Name 为 Credit，Main class 为 com.csu.Credit，Program arguments 和 Working directory 均为事先创建好的“/home/csu/IdeaProjects/Credit”。请特别注意，待处理的数据文件放在什么目录下，Program arguments 的设置就要与那个目录一致。请注意图中的 VM

options 选项，我们所填写的内容是“-Dspark.master=local -Dspark.app.name=Credit -server -XX:PermSize=128M -XX:MaxPermSize=256M”，这意味着将在本地执行程序，并控制不发生内存溢出（可以进行一个实验，将“-server -XX:PermSize=128M -XX:MaxPermSize=256M”删除掉，看看会发生什么）。

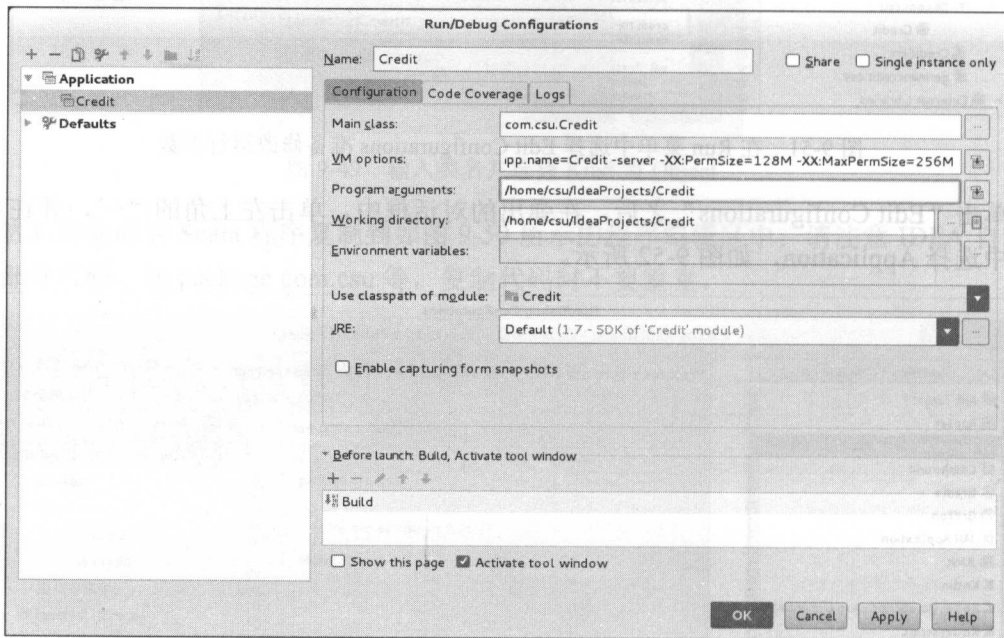


图 9-53 Run/Debug configurations 设置

完成上述运行参数设置，右键单击编辑区中的 Credit 文件（任意位置），选择 Run Credit 即可开始运行程序。下面我们结合代码分析来展示运行结果。

6. 代码分析与执行结果

首先，程序导入机器学习算法的相关包。

```
import org.apache.spark.ml.classification.RandomForestClassifier
import org.apache.spark.ml.evaluation.BinaryClassificationEvaluator
import org.apache.spark.ml.feature.StringIndexer
import org.apache.spark.ml.feature.VectorAssembler
import sqlContext.implicits._
import sqlContext._
import org.apache.spark.ml.tuning.{ ParamGridBuilder, CrossValidator }
import org.apache.spark.ml.{ Pipeline, PipelineStage }
```


接着, 用一个 Scala 的 case 类来定义 Credit 的属性, 对应于 CSV 文件中的一行。

```
// define the Credit Schema
case class Credit(
  creditability: Double, balance: Double, duration: Double, history:
Double, purpose: Double, amount: Double, savings: Double, employment:
Double, instPercent: Double, sexMarried: Double, guarantors: Double,
residenceDuration: Double, assets: Double, age: Double, concCredit:
Double, apartment: Double, credits: Double, occupation: Double,
dependents: Double, hasPhone: Double, foreign: Double )
```

下面的函数用于解析一行数据文件, 将值存入 Credit 类中。类别的索引值减去了 1, 因此起始索引值为 0。

```
// function to create a Credit class from an Array of Double
def parseCredit(line: Array[Double]): Credit = {
  Credit( line(0), line(1) - 1, line(2), line(3), line(4),
line(5), line(6) - 1, line(7) - 1, line(8), line(9) - 1, line(10) -
1, line(11) - 1, line(12) - 1, line(13), line(14) - 1, line(15) - 1, line(16)
- 1, line(17) - 1, line(18) - 1, line(19) - 1, line(20) - 1) }
```

下面的函数用于将字符串 RDD 转换成 Double 类型的 RDD。

```
// function to transform an RDD of Strings into an RDD of Double
def parseRDD(rdd: RDD[String]): RDD[Array[Double]] =
{ rdd.map(_._split(",")).map(_._map(_._toDouble)) }
```

接下来, 是主函数 main。

除了必要的环境设置以外, 主函数首先导入 germancredit.csv 文件中的数据, 存储为一个 String 类型的 RDD。然后对 RDD 做 Map 操作, 将 RDD 中的每个字符串经过 ParseRDD 函数的映射, 转换为一个 Double 类型的数组。紧接着是另一个 Map 操作, 使用 ParseCredit 函数, 将每个 Double 类型的 RDD 转换为 Credit 对象。toDF() 函数则是将 Array[[Credit]] 类型的 RDD 转为一个 Credit 类的 DataFrame。

```
// load the data into a RDD
val creditDF= parseRDD(sc.textFile("germancredit.csv")).
  map(parseCredit).toDF().cache()
creditDF.registerTempTable("credit")
```

程序中, DataFrame 的 printSchema() 函数用于将各个字段含义以树状的形式打印到控制台。


```
// Return the schema of this DataFrame
creditDF.printSchema
```

图 9-54 是在控制台打印的结果。

```
17/03/24 21:48:06 INFO SharedState: Warehouse path is '/home/csu/IdeaProjects/Credit/spark-warehouse'.
17/03/24 21:48:07 INFO SparkSqlParser: Parsing command: credit
root
|-- creditability: double (nullable = false)
|-- balance: double (nullable = false)
|-- duration: double (nullable = false)
|-- history: double (nullable = false)
|-- purpose: double (nullable = false)
|-- amount: double (nullable = false)
|-- savings: double (nullable = false)
|-- employment: double (nullable = false)
|-- instPercent: double (nullable = false)
|-- sexMarried: double (nullable = false)
|-- guarantors: double (nullable = false)
|-- residenceDuration: double (nullable = false)
|-- assets: double (nullable = false)
|-- age: double (nullable = false)
|-- concCredit: double (nullable = false)
|-- apartment: double (nullable = false)
|-- credits: double (nullable = false)
|-- occupation: double (nullable = false)
|-- dependents: double (nullable = false)
|-- hasPhone: double (nullable = false)
|-- foreign: double (nullable = false)

17/03/24 21:48:07 INFO FileInputFormat: Total input paths to process : 1
17/03/24 21:48:07 INFO SparkContext: Starting job: show at Credit.scala:56

All files are up-to-date (a minute ago) 2975:1 LF: UTF-8: [IT] [Q]
```

图 9-54 在控制台打印出的树状形式字段

接着是执行 `creditDF.show`，显示前 20 行 DataFrame，如图 9-55 所示。

```
17/03/24 21:48:08 INFO SparkSqlParser: Parsing command: SELECT creditability, avg(balance) as avgbalance, avg(amount) as avgamt, avg(duration) as avgd
|creditability|balance|duration|history|purpose|amount|savings|employment|instPercent|sexMarried|guarantors|residenceDuration|assets|age|concCredit|a
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|1.0|0.0|18.0|4.0|2.0|1049.0|0.0|1.0|4.0|1.0|0.0|3.0|1.0|21.0|2.0|
|1.0|0.0|9.0|4.0|0.0|2799.0|0.0|2.0|2.0|2.0|0.0|1.0|0.0|36.0|2.0|
|1.0|1.0|12.0|2.0|9.0|841.0|1.0|3.0|2.0|1.0|0.0|3.0|0.0|23.0|2.0|
|1.0|0.0|12.0|4.0|0.0|2122.0|0.0|2.0|3.0|2.0|0.0|1.0|0.0|39.0|2.0|
|1.0|0.0|12.0|4.0|0.0|2171.0|0.0|2.0|4.0|2.0|0.0|3.0|1.0|38.0|0.0|
|1.0|0.0|10.0|4.0|0.0|2241.0|0.0|1.0|1.0|2.0|0.0|2.0|0.0|48.0|2.0|
|1.0|0.0|8.0|4.0|0.0|3398.0|0.0|3.0|1.0|2.0|0.0|3.0|0.0|39.0|2.0|
|1.0|0.0|6.0|4.0|0.0|1361.0|0.0|1.0|2.0|2.0|0.0|3.0|0.0|40.0|2.0|
|1.0|3.0|18.0|4.0|3.0|1098.0|0.0|0.0|4.0|1.0|0.0|3.0|2.0|65.0|2.0|
|1.0|1.0|24.0|2.0|3.0|3758.0|2.0|0.0|1.0|1.0|0.0|3.0|3.0|23.0|2.0|
|1.0|0.0|11.0|4.0|0.0|3905.0|0.0|2.0|2.0|2.0|0.0|1.0|0.0|36.0|2.0|
|1.0|0.0|30.0|4.0|1.0|6187.0|1.0|3.0|1.0|3.0|0.0|3.0|2.0|24.0|2.0|
|1.0|0.0|6.0|4.0|3.0|1957.0|0.0|3.0|1.0|1.0|0.0|3.0|2.0|31.0|2.0|
|1.0|1.0|48.0|3.0|10.0|7582.0|1.0|0.0|2.0|2.0|0.0|3.0|3.0|31.0|2.0|
|1.0|0.0|18.0|2.0|3.0|1936.0|4.0|3.0|2.0|3.0|0.0|3.0|2.0|23.0|2.0|
|1.0|0.0|6.0|2.0|3.0|2647.0|2.0|2.0|2.0|2.0|0.0|2.0|0.0|44.0|2.0|
|1.0|0.0|11.0|4.0|0.0|3939.0|0.0|2.0|1.0|2.0|0.0|1.0|0.0|40.0|2.0|
|1.0|1.0|18.0|2.0|3.0|3213.0|2.0|1.0|1.0|3.0|0.0|2.0|0.0|25.0|2.0|
|1.0|1.0|36.0|4.0|3.0|2337.0|0.0|4.0|4.0|2.0|0.0|3.0|0.0|36.0|2.0|
|1.0|3.0|11.0|4.0|0.0|7228.0|0.0|2.0|1.0|2.0|0.0|3.0|1.0|39.0|2.0|

only showing top 20 rows

All files are up-to-date (4 minutes ago) 39:44 LF: UTF-8: [IT] [Q]
```

图 9-55 creditDF.show 显示前 20 行 DataFrame

DataFrame 初始化之后，程序员就可以用 SQL 命令查询数据了，包括计数、均值、标准差、最小值和最大值。

开发人员还可以用某个表名将 `DataFrame` 注册为一张临时表，然后用 `SQLContext` 提供的方法执行 SQL 命令。例如，用 `sqlContext` 查询语句：

```
sqlContext.sql("SELECT creditability, avg(balance) as avgbalance,
avg(amount) as avgamt, avg(duration) as avgdur FROM credit GROUP BY
creditability ").show
```

其查询结果如图 9-56 所示。

creditability	avgbalance	avgamt	avgdur
0.0	0.9033333333333333	3938.1266666666666	24.86
1.0	1.8657142857142857	2985.442857142857	19.207142857142856

图 9-56 sqlContext 查询结果

提取特征是整个程序的关键。为了构建一个分类模型，首先需要提取对分类最有帮助的特征。在用户信用度数据集里，每条样本用两个类别来标记，1（可信）和 0（不可信）。每个样本的特征包括 21 个字段：其中 1 个标签，表示是否可信，0 或者 1；20 个特征，即“存款”、“期限”、“历史记录”、“目的”、“数额”、“储蓄”、“是否在职”、“分期付款额”、“婚姻”、“担保人”、“居住时间”、“资产”、“年龄”、“历史信用”、“居住公寓”、“贷款”、“职业”、“监护人”、“是否有电话”、“外籍”，通过定义特征数组来存储。

为了在机器学习算法中使用这些特征，这些特征经过了变换，被存入特征向量中，即一组表示各个维度特征值的数值向量。用 `VectorAssembler` 方法将每个维度的特征做变换，返回一个新的 `DataFrame`。

```
//define the feature columns to put in the feature vector
val featureCols = Array("balance", "duration", "history", "purpose",
    "amount", "savings", "employment", "instPercent", "sexMarried",
    "guarantors", "residenceDuration", "assets", "age", "concCredit",
    "apartment", "credits", "occupation", "dependents",
    "hasPhone", "foreign" )
//set the input and output column names
val assembler = new VectorAssembler().setInputCols(featureCols).
    setOutputCol("features")
//return a dataframe with all of the feature columns in a vector
column val df2 = assembler.transform( creditDF)
// the transform method produced a new column: features.
df2.show
```

接着，使用 `StringIndexer` 方法返回一个 `DataFrame`，增加信用度这一列作为标签。

```
// Create a label column with the StringIndexer
val labelIndexer = new StringIndexer().setInputCol
    ("creditability").setOutputCol("label")
val df3 = labelIndexer.fit(df2).transform(df2)
// the transform method produced a new column: label.
df3.show
```

为了训练模型，数据集被分为训练数据和测试数据两个部分，70%的数据用来训练模型，30%的数据用来测试模型。

```
// split the dataframe into training and test data
val splitSeed = 5043
val Array(trainingData, testData) = df3.randomSplit(Array(0.7,
    0.3), splitSeed)
```

接着，程序按照所给参数训练一个随机森林分类器。其中，`maxDepth` 为每棵树的最大深度，增加树的深度可以提高模型的效果，但是会延长训练时间；`NumTrees` 用于设置树的棵数，例如我们这里设置最少为 20，增加决策树棵数，有助于提高预测精度；`maxBins` 为连续特征离散化时选用的最大分桶个数，并且决定每个节点如何分裂；`impurity` 为计算信息增益的指标；`auto` 为在每个节点分裂时是否自动选择参与的特征个数；`seed` 为随机数生成种子。

模型的训练过程就是将输入特征和这些特征对应的样本标签相关联的过程。

```
// create the classifier, set parameters for training
val classifier =
    new RandomForestClassifier().setImpurity("gini").setMaxDepth(3).
        setNumTrees(20).setFeatureSubsetStrategy("auto").setSeed(5043)
// use the random forest classifier to train (fit) the model
val model = classifier.fit(trainingData)
```

一般而言，模型训练结束后需要测试，也就是用测试数据来检验模型效果，也可以视为预测应用。

```
// run the model on test features to get predictions
val predictions = model.transform(testData)
// As you can see, the previous model transform produced a new columns:
    rawPrediction, probability and prediction.predictions.show
```

然后，用 `BinaryClassificationEvaluator` 评估预测的效果，它将预测结果与样本的实际

标签进行比较, 返回一个准确度指标 (ROC 曲线所覆盖的面积)。图 9-57 给出了本书得到的预测准确度, 达到 80.76%。读者实验可能有差异。

```
// create an Evaluator for binary classification, which expects
    two input columns: rawPrediction and label.
    val evaluator = new BinaryClassificationEvaluator().
        setLabelCol("label")
// Evaluates predictions and returns a scalar metric areaUnderROC(
    larger is better).
val accuracy = evaluator.evaluate(predictions)
```

```
accuracy before pipeline fitting0.8075659757616955
```

图 9-57 程序显示的预测准确度

程序接着用管道方式来训练模型。管道 (Pipeline) 采取一种简单方式来比较各种不同组合参数的效果, 这个方法又称为网格搜索法 (Grid Search)。该方法首先设置好待测试的参数, MLLib 会自动完成这些参数的不同组合。管道搭建了一条工作流, 一次性完成了整个模型的调优, 而不是独立对每个参数进行调优。

用 ParamGridBuilder 工具来构建参数网格。

```
// use a ParamGridBuilder to construct a grid of parameters to search over
val paramGrid = new ParamGridBuilder()
    .addGrid(classifier.maxBins, Array(25, 28, 31))
    .addGrid(classifier.maxDepth, Array(4, 6, 8))
    .addGrid(classifier.impurity, Array("entropy", "gini"))
    .build()
```

接着创建并完成一条管道。一条管道由一系列 stage 组成, 每个 stage 相当于一个 Estimator 或是 Transformer。

```
val steps: Array[PipelineStage] = Array(classifier)
val pipeline = new Pipeline().setStages(steps)
```

用 CrossValidator 类来完成模型筛选。CrossValidator 类使用一个 Estimator 类、一组 ParamMaps 类和一个 Evaluator 类。需要注意的是, 使用 CrossValidator 类的开销很大。

```
// Evaluate model on test instances and compute test error
val evaluator = new BinaryClassificationEvaluator()
    .setLabelCol("label")
val cv = new CrossValidator()
    .setEstimator(pipeline)
```

```
.setEvaluator(evaluator)
.setEstimatorParamMaps(paramGrid)
.setNumFolds(10)
```

管道在参数网格上不断地爬行，自动完成模型优化过程。展开来说，对于每个 ParamMap 类，CrossValidator 训练得到一个 Estimator，然后用 Evaluator 来评价结果，然后用最好的 ParamMap 和整个数据集来训练最优的 Estimator。图 9-58 给出了训练管道的执行过程。

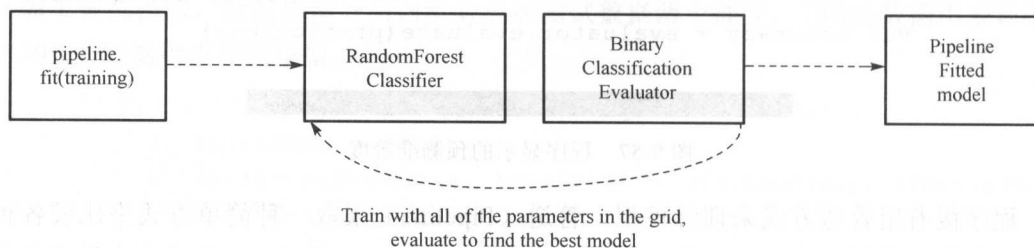


图 9-58 训练管道

```
// When fit is called, the stages are executed in order.
// Fit will run cross-validation, and choose the best set of parameters
// The fitted model from a Pipeline is an PipelineModel, which
// consists of fitted models and transformers
val pipelineFittedModel = cv.fit(trainingData)
```

至此，程序可以用管道训练得到的最优模型进行预测应用了。将预测结果与标签做比较，预测结果取得了 81.67% 的准确率，相比之前 80.76% 的准确率有所提高，如图 9-59 所示。

```
accuracy after pipeline fitting0.8167402876883011
RandomForestClassificationModel with 60 trees
```

图 9-59 管道训练得到 81.67% 的准确率

```
// call tranform to make predictions on test data. The fitted
// model will use the best model found
val predictions = pipelineFittedModel.transform(testData)
val accuracy = evaluator.evaluate(predictions)
val rm2 = new RegressionMetrics(
  predictions.select("prediction", "label").rdd.map(x =>
    (x(0).asInstanceOf[Double], x(1).asInstanceOf[Double])))
println("MSE: " + rm2.meanSquaredError)
println("MAE: " + rm2.meanAbsoluteError)
println("RMSE Squared: " + rm2.rootMeanSquaredError)
```

```
println("R Squared: " + rm2.r2)
println("Explained Variance: " + rm2.explainedVariance + "\n")
```

上面详细描述了如何用 Apache Spark MLlib 的机器学习随机森林算法和机器学习管道来解决分类问题。开发人员完全可以参考上述方法实现一个实际的风险预测系统。

9.3 本章小结

本章首先介绍了 Spark 的基本原理和架构，分析了 Spark 的几个关键概念，特别是 RDD、DAG 和 DataFrame。

本章重点介绍了 Spark 的基本应用，通过编写计算圆周率 Pi、基于随机森林模型的贷款风险预测 Scala 程序，演示了在集成开发环境 IDEA 中编写 Spark 程序和进行必要参数设置的方法，使得读者感受了 Spark 对分布式并行计算的支持。

第三篇

数据处理与项目开发

第 10 章

交互式数据处理

10

从本章开始，我们把重点转移到大数据处理本身上面来。归根结底，Hadoop 及其组件都是用来分析大数据的。本章首先展示如何在 Hadoop 平台上进行交互式数据处理，我们主要以 Hive 组件为基本工具，介绍相关方法的运用。

10.1 数据预处理

为了保证实践的真实性，本章为读者提供了一个较大的数据文件，这就是 sogou.500w.utf8。该文件是大数据领域很有名的一个供研究用的数据文件，内容是 sogou 网络访问日志数据，该文件被众多研究和开发人员所采用。

10.1.1 查看数据

请读者到第 10 章软件资源文件夹内找到 sogou.500w.utf8 文件，将其复制到 Master 节点的“/home/csu/resources/”目录（或者读者自己的任意目录）下。以下的大部分操作均围绕该数据文件进行。

首先进入实验数据所在文件夹，即执行“cd /home/csu/resources”，然后通过 less 命令查看 sogou.500w.utf8 文件内容，如图 10-1 所示。

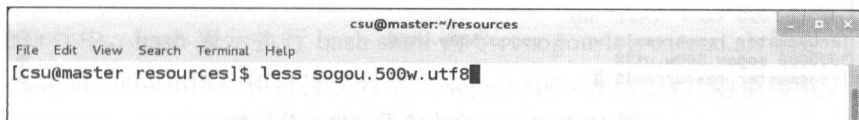


图 10-1 准备用 less 命令查看数据文件的内容

Linux 中的 less 命令主要用来浏览文件内容, 与 more 命令的用法相似。不同于 more 命令的地方是, less 可回滚浏览已经看过的部分, 所以, less 的用法比 more 更灵活。在使用 more 的时候, 用户不能向前面翻, 只能往后面看, 但是如果使用 less 命令, 就可以配合 pageup 和 pagedown 等按键的功能, 来回翻看文件, 更容易浏览一个文件的内容。除此之外, 利用 less 命令还可以进行向上和向下的搜索。

图 10-2 给出了查看结果。读者可以运用刚才介绍的方法浏览文件。



图 10-2 查看结果

现在我们主要关心的还是文件内容。sogou.500w.utf8 是一种 utf-8 格式的文件, 这是一种文本类型的格式文件, 它针对每个文字进行相应的编码, 用双字节存储汉字, 而英文单词和符号则用单字节存储。sogou.500w.utf8 文件很大, 有 500 万条记录, 每一条记录包括了访问时间、用户 ID、查询词、返回结果排序、用户单击的顺序号、用户单击的 URL, 共计 6 个字段。字段与字段之间是通过一个 “\t” (Tab) 分割的。由于 sogou.500w.utf8 有 500 万条记录, 所以我们这里只是查看一下部分内容, 目的是让读者看看大数据文件到底是什么样子, 没有必要从头看到尾。要终止上述查看, 先按一下 esc 键, 回车, 再按 q 键即可。

下面, 我们再通过执行 wc 命令, 查看一下文件的总行数, 如图 10-3 所示。

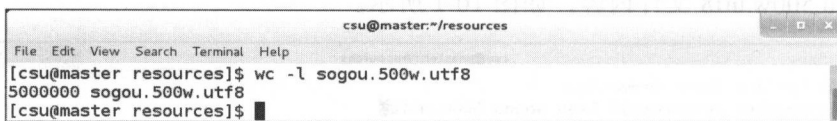


图 10-3 用 wc 命令查看文件的总行数

回车后系统显示 “5000000 sogou.500w.utf8”。可见文件确实有 500 万行。Linux 中 wc

命令的功能为统计指定文件中的行数、字数、字节数，并将统计结果显示输出。参数-l 表示统计行数，-w 表示统计字数，-c 表示统计字节数。

另一个有用的操作是 head 命令。有时候，用户希望截取文件的部分数据，head 命令就可以完成这个任务，如图 10-4 所示。

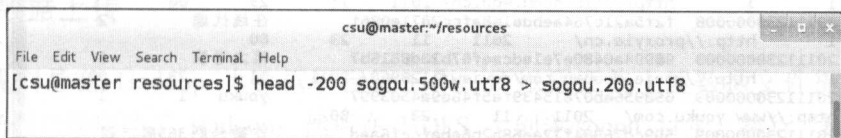


图 10-4 用 head 截取文件

回车执行后，我们得到了一个 200 行的新数据文件 sogou.200.utf8（文件名由用户自己决定，可以是任意的名字，如 sogou.demo）。读者也可以查看一下新文件的内容或行数。

10.1.2 数据扩展

很多时候用户希望扩展现有文件，例如增加新的字段，以便容纳更多的内容。下面我们就来扩展 sogou.500w.utf8 文件，增加年、月、日、小时这样 4 个新字段。这样，扩展后的文件就有 10 个字段了。

请读者首先到本章软件资源文件夹中，把一个名为 sogou-log-extend.sh 文件复制到 Master 节点的“/home/csu/”目录下。该文件是一个 Linux shell 脚本文件，里面的内容是扩展文件字段的命令。

把 sogou-log-extend.sh 文件复制到位后，请执行如图 10-5 所示的命令。

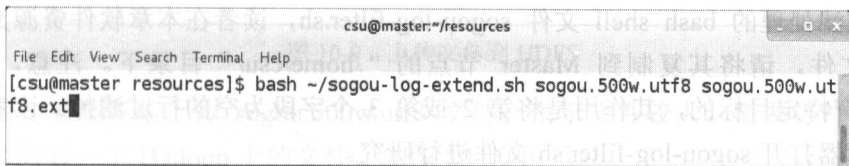


图 10-5 数据文件的扩展

上述命令行中，bash 表示执行 bash shell 命令，sogou-log-extend.sh 就是执行对象，而源文件名（sogou.500w.utf8）和目标文件名（sogou.500w.utf8.ext）则是参数。回车后，系统会执行一段时间，最后得到一个含有 10 个字段的新数据文件 sogou.500w.utf8.ext。不妨用 less 命令查看一下其内容，如图 10-6 所示。



图 10-6 用 less 命令查看扩展后的数据文件

可以看到，每一行都增加了 4 个新字段，分别是年、月、日、小时，其内容是从第一个字段分离出来的。例如，第一个字段是“2011112300000010”，则新增加的 4 个字段就是“2011、11、23、00”。

显然，我们做这种扩展，目的是为了在将来进行统计分析的时候，操作更加方便快捷。

10.1.3 数据过滤

有时候，我们需要过滤数据文件。例如，对某些字段为空的行，我们可能想将这些行过滤掉，即从文件中删除掉。我们来处理一下 `sogou.500w.utf8.ext` 文件。同样，我们需要一个用于过滤处理的 `bash shell` 文件 `sogou-log-filter.sh`，读者在本章软件资源文件夹中可以找到该文件，请将其复制到 Master 节点的“/home/csu/”目录下。注意，`sogou-log-filter.sh` 是有特定目标的，其作用是将第 2 或第 3 个字段为空的行过滤掉。读者可以用任何文本编辑器打开 `sogou-log-filter.sh` 文件进行研究。

把 `sogou-log-filter.sh` 文件复制到位后，请执行如图 10-7 所示的命令。

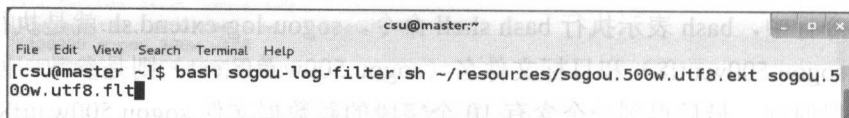


图 10-7 对数据文件进行过滤处理

回车后，系统开始进行过滤。这里需要耐心等待一下，因为处理 500 万条记录还是需要一点时间的。实际上 `sogou.500w.utf8.ext` 文件并没有字段为空的行，所以读者会看到过滤后得到的 `sogou.500w.utf8.flit` 文件与原来的 `sogou.500w.utf8.ext` 文件完全一样。我们这里主要还是为了给出一个过滤处理的示范。

10.1.4 数据上传

毕竟我们是在 Hadoop 大数据平台上工作，所以需要将上述数据文件提交到 HDFS 上。首先确保已经启动了 Hadoop；然后在 HDFS 上创建 `/sogou` 目录，执行“`hadoop fs -mkdir /sogou`”；最后创建 20161202 子目录，命令是“`hadoop fs -mkdir /sogou/20161202`”，如图 10-8 所示。

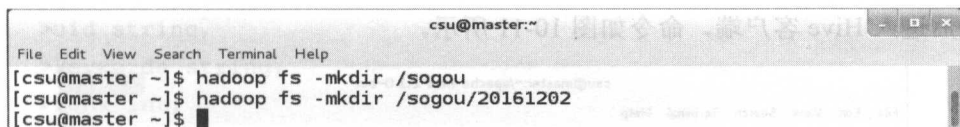


图 10-8 在 Hadoop 上创建数据目录

注意，如果想创建“`/sogou/20161202/`”这样的两级目录，必须先创建“`/sogou`”。换言之，在没有创建上级目录之前，不能使用像“`hadoop fs -mkdir /sogou/20161202`”这样的命令创建下一级目录。

有了“`/sogou/20161202/`”目录，我们就可以将 `sogou.500w.utf8` 上传到 HDFS 里了，命令如图 10-9 所示。

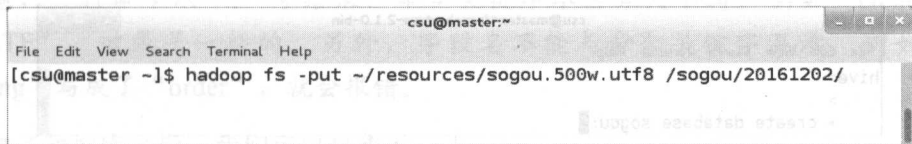


图 10-9 上传文件到 HDFS

回车后，系统开始上传 `sogou.500w.utf8` 文件。由于文件较大，需要等待片刻。传送完毕，可以查看一下 Hadoop 上的文件，如图 10-10 所示。

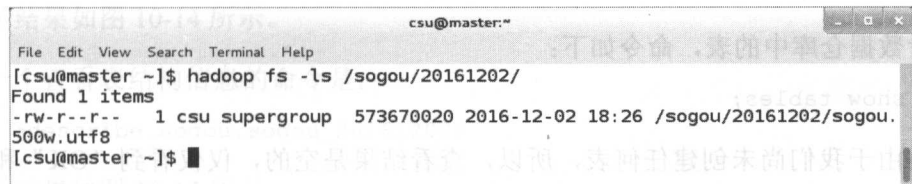


图 10-10 查看上传的文件列表

同理，我们也把过滤后得到的 `sogou.500w.utf8.flt` 文件上传到 HDFS 中，分别执行如下三条命令即可。

```
hadoop fs -mkdir /sogou_ext
hadoop fs -mkdir /sogou_ext/20161202
hadoop fs -put ~/resources/sogou.500w.utf8.flt /sogou_ext/20161202/
```

10.2 创建数据仓库

我们的目标是在 Hive 中创建数据仓库，以便利用 Hive 的查询功能实现交互式应用。所以，接下来我们在 Hive 客户端进行操作。注意，要确保 Hadoop 和 MySQL 服务已经启动，然后进入 Hive 客户端，命令如图 10-11 所示。

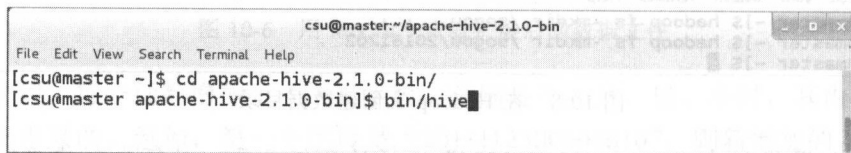


图 10-11 准备进入 Hive Shell

10.2.1 创建 Hive 数据仓库的基本命令

我们先来执行如下命令，尝试创建一个数据仓库，如图 10-12 所示。

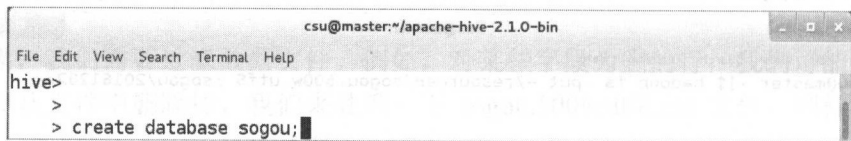


图 10-12 在 Hive 中创建数据仓库

创建数据仓库以后，可以打开数据仓库，命令如下：

```
use sogou;
```

查看数据仓库中的表，命令如下：

```
show tables;
```

目前由于我们尚未创建任何表，所以，查看结果是空的，仅仅看到“OK”和“Time taken: 0.032 seconds”的提示，如图 10-13 所示。

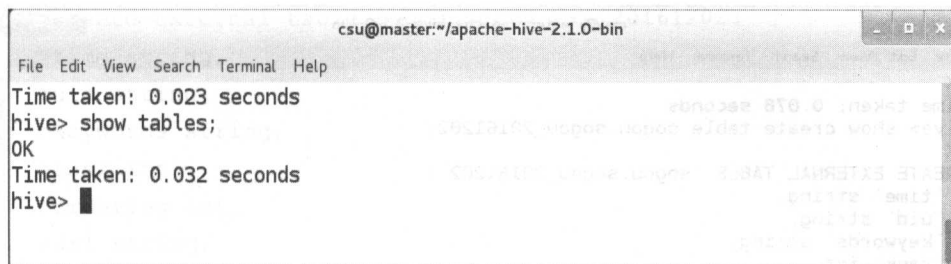


图 10-13 查看数据仓库中的表

下面，我们来创建一个外部表，命令如下：

```

>create external table sogou.sogou_20161202(
>time string,
>uid string,
>keywords string,
>rank int,
>ordering int,
>url string)
>comment 'This is the sogou search data of one day'
>row format delimited
>fields terminated by '\\t'
>stored as textfile
>location '/sogou/20161202';

```

注意，一行命令以“;”号结束，命令关键字不区分大小写，例如“create”写成“CREATE”，效果是一样的。另外，字段名不能与命令关键字混淆，例如，如果“ordering”写成了“order”，就会报错。

创建了数据库之后，我们可以来查看一下。

查看表结构的命令是：

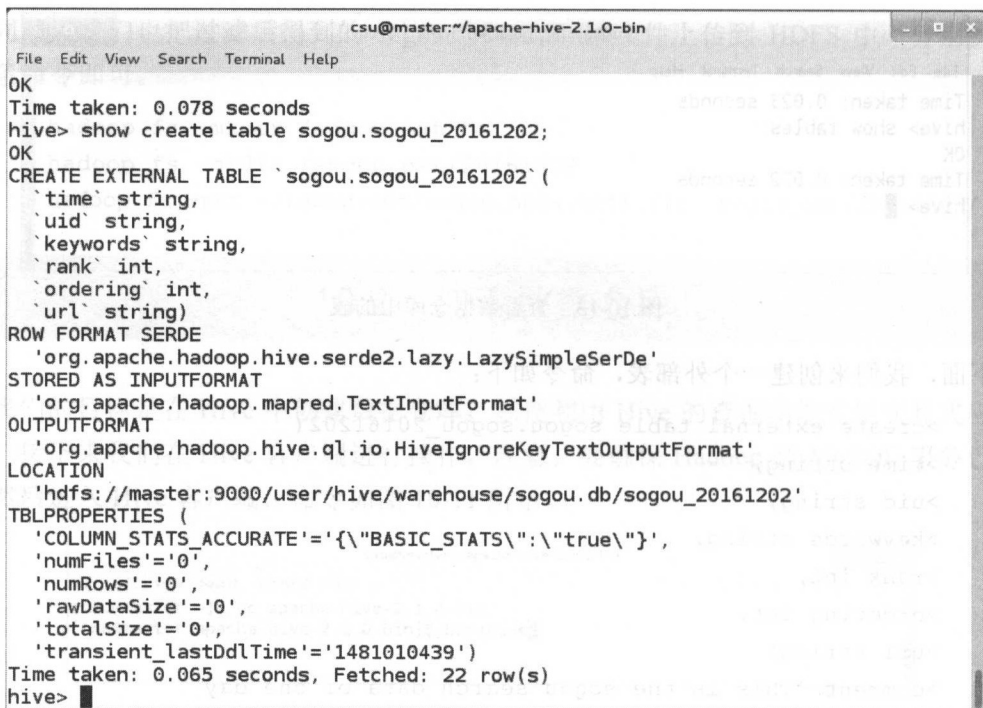
```
>show create table sogou.sogou_20161202;
```

显示结果如图 10-14 所示。

另一个查看表结构信息的命令是：

```
>describe sogou.sogou_20161202;
```

显示结果如图 10-15 所示。

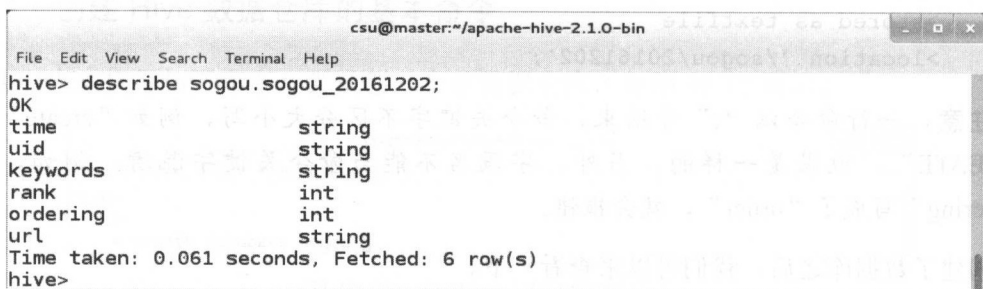


```

csu@master:~/apache-hive-2.1.0-bin
File Edit View Search Terminal Help
OK
Time taken: 0.078 seconds
hive> show create table sogou.sogou_20161202;
OK
CREATE EXTERNAL TABLE `sogou.sogou_20161202` (
  `time` string,
  `uid` string,
  `keywords` string,
  `rank` int,
  `ordering` int,
  `url` string)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'
STORED AS INPUTFORMAT
  'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
  'hdfs://master:9000/user/hive/warehouse/sogou.db/sogou_20161202'
TBLPROPERTIES (
  'COLUMN_STATS_ACCURATE'='{\"BASIC_STATS\": \"true\"}',
  'numFiles'='0',
  'numRows'='0',
  'rawDataSize'='0',
  'totalSize'='0',
  'transient_lastDdlTime'='1481010439')
Time taken: 0.065 seconds, Fetched: 22 row(s)
hive>

```

图 10-14 用 show 命令查看表结构信息



```

csu@master:~/apache-hive-2.1.0-bin
File Edit View Search Terminal Help
hive> describe sogou.sogou_20161202;
OK
time                string
uid                 string
keywords            string
rank                int
ordering            int
url                 string
Time taken: 0.061 seconds, Fetched: 6 row(s)
hive>

```

图 10-15 用 describe 命令查看表结构信息

要删除已经创建的表，使用的命令如下：

```
>drop table sogou.sogou_20161202;
```

10.2.2 创建 Hive 分区表

上面实际上是一个练习。下面我们正式创建一个外部表，该表包含了扩展字段（即 year、month、day、hour），命令如下：

```

>create external table sogou.sogou_ext_20161202(
>time string,
>uid string,
>keywords string,
>rank int,
>ordering int,
>url string,
>year int,
>month int,
>day int,
>hour int)
>comment 'This is the sogou search data extend'
>row format delimited
>fields terminated by '\t'
>stored as textfile
>location '/sogou_ext/20161202';

```

上述命令中，特别要注意，location 后面的“/sogou_ext/20161202”就是我们在前面创建的 HDFS 目录，并且已经上传了 sogou.500w.utf8.flit 文件。

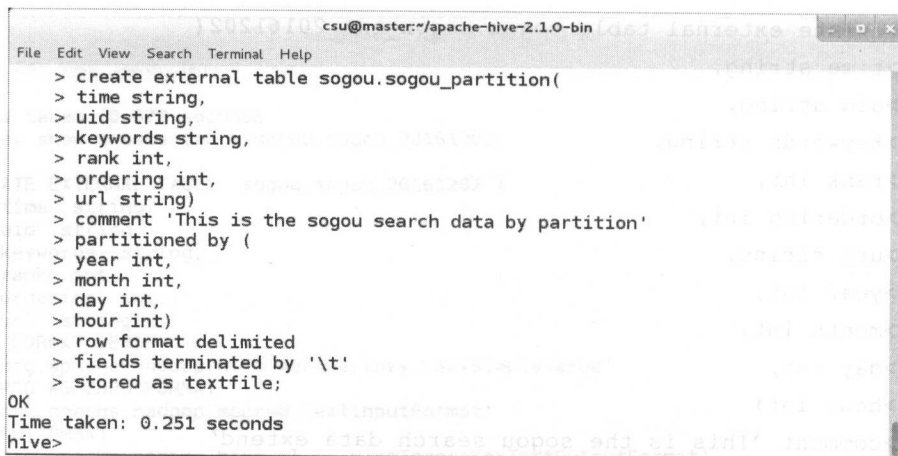
接着创建带分区的表，命令如下：

```

>create external table sogou.sogou_partition(
>time string,
>uid string,
>keywords string,
>rank int,
>ordering int,
>url string)
>partitioned by (
>year int,
>month int,
>day int,
>hour int)
>row format delimited
>fields terminated by '\t'
>stored as textfile;

```

为清楚起见，我们给出执行上述命令的截图，如图 10-16 所示。



```

csu@master:~/apache-hive-2.1.0-bin
File Edit View Search Terminal Help
> create external table sogou.sogou_partition(
> time string,
> uid string,
> keywords string,
> rank int,
> ordering int,
> url string)
> comment 'This is the sogou search data by partition'
> partitioned by (
> year int,
> month int,
> day int,
> hour int)
> row format delimited
> fields terminated by '\t'
> stored as textfile;
OK
Time taken: 0.251 seconds
hive>

```

图 10-16 创建带分区的表

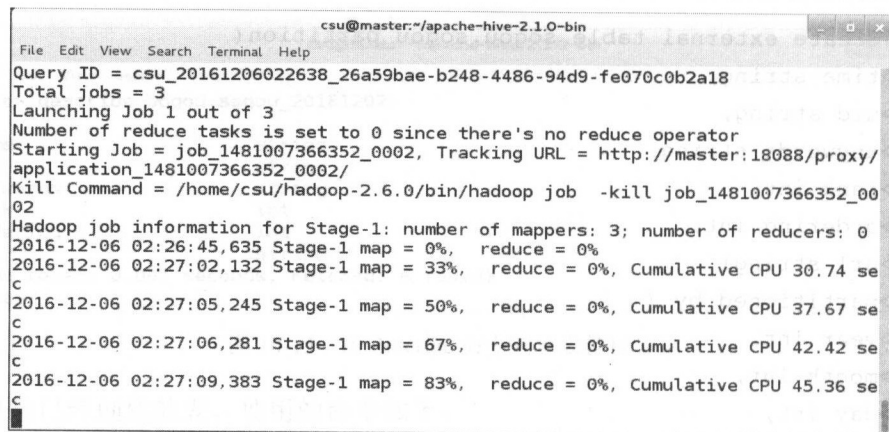
最后，我们来往数据库中导入数据，命令是：

```

>set hive.exec.dynamic.partition.mode=nonstrict;
>insert overwrite table sogou.sogou_partition partition(year,month,day,hour)
    select * from sogou.sogou_ext_20161202;

```

回车后，系统滚动显示执行状态信息，如图 10-17 所示。



```

csu@master:~/apache-hive-2.1.0-bin
File Edit View Search Terminal Help
Query ID = csu_20161206022638_26a59bae-b248-4486-94d9-fe070c0b2a18
Total jobs = 3
Launching Job 1 out of 3
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1481007366352_0002, Tracking URL = http://master:18088/proxy/
application_1481007366352_0002/
Kill Command = /home/csu/hadoop-2.6.0/bin/hadoop job -kill job_1481007366352_00
02
Hadoop job information for Stage-1: number of mappers: 3; number of reducers: 0
2016-12-06 02:26:45,635 Stage-1 map = 0%, reduce = 0%
2016-12-06 02:27:02,132 Stage-1 map = 33%, reduce = 0%, Cumulative CPU 30.74 se
c
2016-12-06 02:27:05,245 Stage-1 map = 50%, reduce = 0%, Cumulative CPU 37.67 se
c
2016-12-06 02:27:06,281 Stage-1 map = 67%, reduce = 0%, Cumulative CPU 42.42 se
c
2016-12-06 02:27:09,383 Stage-1 map = 83%, reduce = 0%, Cumulative CPU 45.36 se
c

```

图 10-17 执行导入数据的过程

最后，我们查询一下导入的数据，命令如下：

```
>select * from sogou_ext_20161202 limit 10;
```

查询结果如图 10-18 所示。

```

csu@master:~/apache-hive-2.1.0-bin
File Edit View Search Terminal Help
hive> select * from sogou.sogou_ext_20161202 limit 10;
OK
20111230000005 57375476989eea12893c0c3811607bcf 奇艺高清 1 1
http://www.qiyi.com/ 2011 11 23 0
20111230000005 66c5bb7774e31d0a22278249b26bc83a 凡人修仙传 3 1
http://www.booksky.org/BookDetail.aspx?BookID=1050804&Level=1 2011 11 2
3 0
20111230000007 b97920521c78de70ac38e3713f524b50 本本联盟 1 1
http://www.bblianmeng.com/ 2011 11 23 0
20111230000008 6961d0c97fe93701fc9c0d861d096cd9 华南师范大学图书馆 1
1 http://lib.scnu.edu.cn/ 2011 11 23 23
20111230000008 f2f5a21c764aebde1e8afcc2871e086f 在线代理 2 1
http://proxyie.cn/ 2011 11 23 0
20111230000009 96994a0480e7e1edcaef67b20d8816b7 伟大导演 1 1
http://movie.douban.com/review/1128960/ 2011 11 23 0
20111230000009 698956eb07815439fe5f46e9a4503997 youku 1 1 h
ttp://www.youku.com/ 2011 11 23 0
20111230000009 599cd26984f72ee68b2b6ebefccf6aed 安徽合肥365房产网 1
1 http://hf.house365.com/ 2011 11 23 23
20111230000010 f577230df7b6c532837cd16ab731f874 哈萨克网址大全 1 1
http://www.kz321.com/ 2011 11 23 0
20111230000010 285f88780dd0659f5fc8acc7cc4949f2 IQ数码 1 1 h
ttp://www.iqshuma.com/ 2011 11 23 0
Time taken: 0.103 seconds, Fetched: 10 row(s)
hive>

```

图 10-18 查询导入的数据

其他查询命令还有：

```

>select url from sogou_ext_20161202 limit 10;
>select * from sogou_ext_20161202 where uid=' 96994a0480e7e1edcaef67b20d8816b7';

```

10.3 数据分析

上面我们建立了 sogou 日志数据仓库，接下来就可以进行一系列的分析操作了。

10.3.1 基本统计

1. 统计总记录数

命令是：

```
>select count(*) from sogou_ext_20161202;
```

执行结果如图 10-19 所示。

为节省篇幅，下面的操作仅仅给出命令，不再展示结果。


```

csu@master:~/apache-hive-2.1.0-bin
File Edit View Search Terminal Help
07
Hadoop job information for Stage-1: number of mappers: 3; number of reducers: 1
2016-12-06 04:33:25,324 Stage-1 map = 0%, reduce = 0%
2016-12-06 04:33:32,665 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 8.97 se
c
2016-12-06 04:33:37,914 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 10.23
sec
MapReduce Total cumulative CPU time: 10 seconds 230 msec
Ended Job = job_1481007366352_0007
MapReduce Jobs Launched:
Stage-Stage-1: Map: 3 Reduce: 1 Cumulative CPU: 10.23 sec HDFS Read: 643703
669 HDFS Write: 107 SUCCESS
Total MapReduce CPU Time Spent: 10 seconds 230 msec
OK
50000000
Time taken: 18.383 seconds, Fetched: 1 row(s)
hive>

```

图 10-19 统计总记录数

2. 统计非空记录数

```
>select count(*) from sogou_ext_20161202 where keywords is not
null and keywords!='';
```

统计独立 uid 总数:

```
>select count(distinct(uid)) from sogou.sogou_ext_20161202;
```

3. 关键词分析

关键词长度统计:

```
>select avg(a.cnt) from (select size(split(keywords,'\s+')) as
cnt from sogou.sogou_ext_20161202) a;
```

频度排名 (即频度最高的前 20 个词):

```
>select keywords, count(*) as cnt from sogou.sogou_ext_20161202
group by keywords order by cnt desc limit 20;
```

4. uid 分析

查询次数的分布:

```
>select sum(if(uids.cnt=1,1,0)), sum(if(uids.cnt=2,1,0)), sum
(if(uids.cnt=3,1,0)), sum(if(uids.cnt>3,1,0)) from (select uid, count(*)
as cnt from sogou.sogou_ext_20161202 group by uid) uids;
```

平均查询次数:

```
>select sum(a.cnt)/count(a.uid) from (select uid,count(*) as cnt
from sogou_ext_20161202 group by uid) a;
```

查询次数大于 2 次的用户总数:

```
>select count(a.cnt) from (select uid, count(*) as cnt from
sogou.sogou_ext_20161202 group by uid having cnt > 2 ) a;
```

查询次数大于 2 次的用户占比:

```
>select count(distinct (uid)) form sogou.sogou_ext_20161202;
>select count(a.cnt) from (select uid, count(*) as cnt from
sogou.sogou_ext_20161202 group by uid having cnt>2) a;
```

设上述两项操作的结果分别是 A、B，则查询次数大于 2 次的用户的占比等于 B/A。

查询次数大于 2 次的数据展示:

```
>select b.* from
>(select uid,count(*) as cnt from sogou.sogou_ext_20161202 group
by uid having cnt>2) a
>join sogou.sogou_ext_20161202 b on a.uid=b.uid
>limit 50;
```

10.3.2 用户行为分析

1. 点击次数与 rank 之间的关系

下面我们来计算 rank 在 10 以内的点击次数占比。

首先，我们执行:

```
>select count(*) from sogou.sogou_ext_20161202 where rank<11;
```

结果如图 10-20 所示。

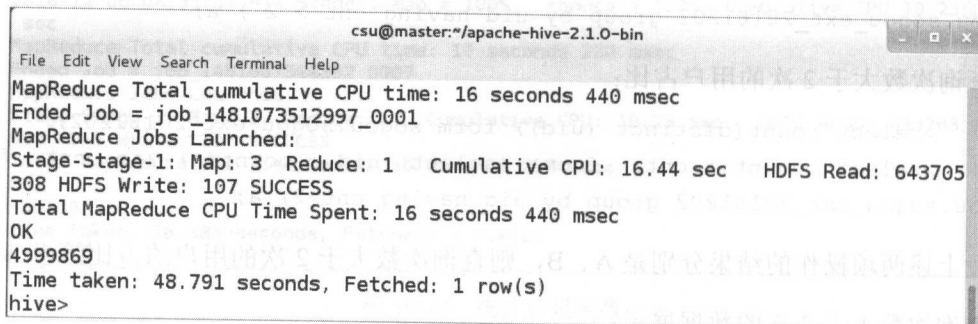
然后执行:

```
>select count(*) from sogou.sogou_ext_20161202;
```

这条命令的执行结果当然是 5000000。

设上述计算的结果分别是 A、B，显然所需要的结果就是 A/B，即 4999869/5000000，等于 0.9999738。这个结果很有意思。我们知道，用户上网查询往往只会浏览搜索引擎返回结果的前 10 个项目，也就是位于第一页的内容。这个用户行为说明，尽管搜索引擎的返

回数十分庞大，但是真正可能被用户关注的内容往往很少，只有排在最前面的很小部分会被用户浏览到，所以，传统的基于全部返回值计算的查全率、查准率的评价方式已经不适应网络信息检索的评价。正确的评价方式，应该强调评价指标中有关最靠前的结果与用户查询需求之间的相关性。



```

csu@master:~/apache-hive-2.1.0-bin
File Edit View Search Terminal Help
MapReduce Total cumulative CPU time: 16 seconds 440 msec
Ended Job = job_1481073512997_0001
MapReduce Jobs Launched:
Stage-Stage-1: Map: 3 Reduce: 1 Cumulative CPU: 16.44 sec HDFS Read: 643705
308 HDFS Write: 107 SUCCESS
Total MapReduce CPU Time Spent: 16 seconds 440 msec
OK
4999869
Time taken: 48.791 seconds, Fetched: 1 row(s)
hive>

```

图 10-20 排名前十链接的单击次数

我们再来研究直接通过输入 URL 进行查询的占比：

```

>select count(*) from sogou.sogou_ext_20161202 where keywords like
'www%';
>select count(*) from sogou.sogou_ext_20161202;

```

上述两次计算结果分别是 A、B，则 A/B 即为所需要的结果，实际结果是 73979/5000000，等于 0.0147958。这个比例是很低的，说明绝大部分用户不会采用 URL 进行查询。想一想也很自然，如果用户知道了 URL，完全可以直接在浏览器地址栏输入进行查询，没有必要再通过搜索引擎重复一遍。

另外，我们还可以计算，在通过 URL 进行的查询中，用户点击了其输入的 URL 网址的次数，并计算占比。

```

>select sum(if(instr(url,keywords)>0,1,0)) from (select * from
sogou.sogou_ext_20161202 where keywords like 'www%') a;

```

设这次计算的结果是 C (27561)，则 C/A (27561/73979=0.37255167) 即我们关心的结果。从这个比例可以看出，有 37%的用户（应该说是很大一部分）提交了 URL 进行查询，并且继续点击了查询的结果。这可能是由于用户没有记全 URL 等原因，而想借助搜索引擎来找到自己想要的网址。因此，这个分析结果就提示我们，搜索引擎在处理这一部分查询请求的时候，一个可能比较理想的改进方式就是，首先把相关的完整 URL 返回给用户，这样就有较大可能改善用户的查询体验，满足用户需求。

2. 个性化行为分析

例如, 如果想知道搜索过“仙剑奇侠传”且次数大于 3 的 uid, 命令如下。

```
>select uid,count(*) as cnt from sogou.sogou_ext_20161202 where
keywords='仙剑奇侠传' group by uid having cnt >3;
```

计算结果如图 10-21 所示。可以看到, 有两人满足条件, 搜索次数分别是 6 和 5。

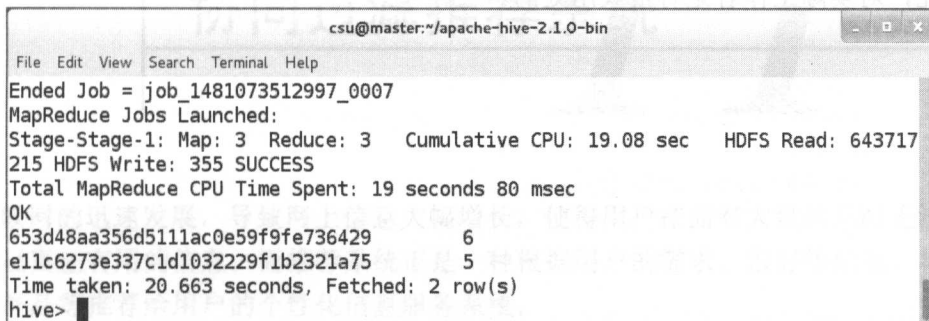


图 10-21 查询搜索过“仙剑奇侠传”且次数大于 3 的 uid

10.3.3 实时数据

实际应用中, 为了实时地显示当天搜索引擎的搜索数据, 我们首先需要创建一些临时表, 然后在一天结束后对数据进行处理, 并将数据插入到临时表中, 供显示部分展示。

1. 创建临时表

```
>create table sogou.uid_cnt(uid string, cnt int)
>comment 'This is the sogou search data of one day'
>row format delimited
>fields terminated by '\t'
>stored as textfile;
```

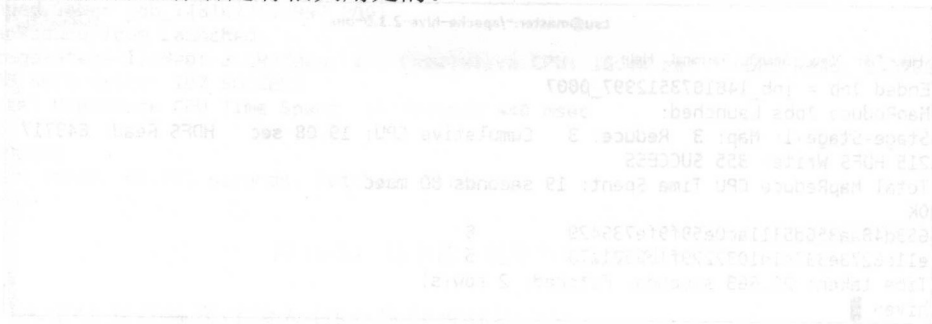
2. 插入数据

```
>insert overwrite table sogou.uid_cnt select uid, count(*) as cnt
>from sogou.sogou_ext_20161202 group by uid;
```

这样前端开发人员就可以访问该临时表, 并将数据展示出来, 其展示方式可以根据实际需要设计, 如表格、统计图等。

10.4 本章小结

本章介绍了如何利用 Hive 进行大数据处理和分析。Hive 是建立在 Hadoop MapReduce 基础上的数据仓库工具，用户只要借助 SQL 语句，即可完成很多处理和分析，因此，对实际工作者是有很大大用处的。



10.3.3 数据仓库

互联网的迅速发展,导致网上信息大幅增长,使得用户在面对大量信息时无法从中获得对自己真正有用的信息。而推荐系统正是一种根据用户的需求、偏好等信息,将用户感兴趣的产品等推荐给用户的个性化信息服务系统。

与搜索引擎相比,推荐系统通过研究用户的兴趣偏好,进行个性化计算,由系统发现用户的兴趣点,从而引导用户发现自己的需求。一个好的推荐系统不仅能为用户提供个性化的服务,还能和用户之间建立密切关系,让用户对推荐产生信赖。

推荐系统现已广泛应用于很多领域,其中最典型并具有良好的发展前景的领域就是电子商务。同时,推荐系统也得到学术界的高度关注,并逐步发展成一门独立的学科。

11.1 推荐算法概述

推荐系统的任务就是在用户无法准确描述自己的需求时,帮助用户发现对自己有价值的信息,并让信息能够展现在对它感兴趣的人群中,从而实现信息提供商与用户的双赢。推荐方法总体上可以分为基于人口统计学的推荐、基于内容的推荐、基于协同过滤的推荐、基于知识的推荐,以及组合推荐几类。下文主要介绍前三种。

11.1.1 基于人口统计学的推荐

基于人口统计学的推荐(Demographic-Based Recommendation)是最为简单的一种推荐算法,它根据系统用户的基本信息来发现用户之间的相关程度,然后将相似用户喜爱的其他物品推荐给当前用户,如图 11-1 所示。

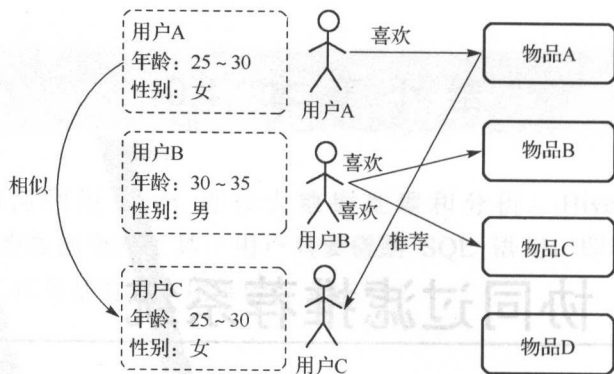


图 11-1 基于人口统计学的推荐系统原理

系统首先会根据用户的属性建模，如用户的年龄、性别、兴趣等信息，然后根据这些特征计算用户间的相似度。例如，系统通过计算，发现用户 A 和 C 比较相似，于是就把 A 喜欢的物品 A 推荐给用户 C。

这种方法的优点是，不依赖于物品的属性，因此其他领域的问题都可无缝接入；同时，系统并不一定需要用户对物品的历史评价信息（即没有新用户冷启动问题）。该算法的主要挑战是获取用户的个人信息，如显性地请用户直接提供年龄、性别、住址之类的信息。如果这样直接获取用户信息不行，也可以通过分析技术，间接提取用户留在系统中的交互信息，建立用户的人口统计数据库。

11.1.2 基于内容的推荐

基于内容的推荐是一种既依赖物品属性，也依赖用户对物品的历史评价信息的推荐方法。换言之，基于内容的推荐，其信息来源有两个方面，一是物品属性，二是用户的历史评价（偏好）数据，而基于人口统计学的推荐只需要用户信息，不依赖物品属性。

系统首先对物品（如图 11-2 中以电影为例）的属性进行建模，图中用类型作为属性。当然，在实际应用中，不仅需要考虑电影的类型，还可以考虑演员、导演等更多信息。接着，系统通过相似度计算，发现电影 A 和 C 相似度较高，因为他们都属于爱情类。系统同时还发现用户 A 喜欢电影 A，由此得出结论，用户 A 很可能对电影 C 也感兴趣，于是将电影 C 推荐给 A。

基于内容的推荐方法的优点是，通过对物品属性维度的增加，可以获得更好的推荐精度；同时，系统还可以对用户的兴趣进行建模。缺点是，由于物品属性毕竟是有限的，因此很难进一步扩展更多的属性数据，而且物品相似度的衡量标准只考虑到了物品本身，有

一定的片面性。此外，该方法还需要用户对物品的历史评价数据，这就使得新用户因为没有历史评价数据，导致系统无法获知其与某物品的联系，出现新用户冷启动的问题。

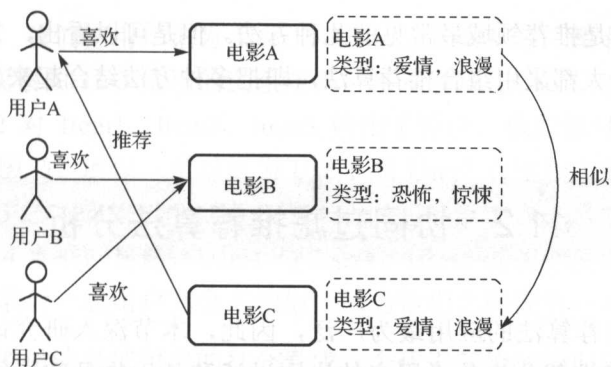


图 11-2 基于内容的推荐系统原理

11.1.3 协同过滤推荐

协同过滤是利用集体智慧进行推荐的一种典型方法。

想象一下，如果你现在想去看电影，但不知道具体看哪一部电影，你会怎么做？大部分人会问问周围的朋友，看看最近有什么好看的电影推荐一下。关键是，我们一般会向兴趣比较类似的朋友咨询，从而获得比较精准的推荐。这就是协同过滤的基本思想。

协同过滤（**Collaborative Filtering, CF**），又称为社会关系过滤，是指利用某个兴趣相投、拥有共同经验之群体的喜好来推荐感兴趣物品的方法，并且允许用户通过合作机制对推荐结果进行反馈（如评分）并记录下来以达到过滤的目的。

协同过滤是推荐算法中最常用的一种方法，可分为基于用户的协同过滤和基于物品的协同过滤。

协同过滤的优势在于，它不需要对物品或者用户进行严格的建模（这是与前两种算法的主要区别），而且不要求物品的描述是机器可理解的，它只要建立用户与物品的某种关系（例如评价关系）矩阵，就足以支撑推荐系统的运行，所以这种方法也是领域无关的。而且，这种方法计算出来的推荐是开放的，可以共用他人的经验，往往能够向用户推荐新颖的物品，支持用户发现潜在的兴趣偏好。

协同过滤的缺点是，方法的核心是基于历史数据，所以对新物品和新用户都有“冷启动”的问题，而且推荐的效果依赖于用户历史偏好数据的多少和准确性。此外，在大部分的实现中，用户历史偏好是用稀疏矩阵存储的，而稀疏矩阵的计算是一个挑战，包括可能

出现少部分人的错误偏好会对推荐的准确度有很大的影响，导致对一些有特殊品味的用户不能给予很好的推荐。

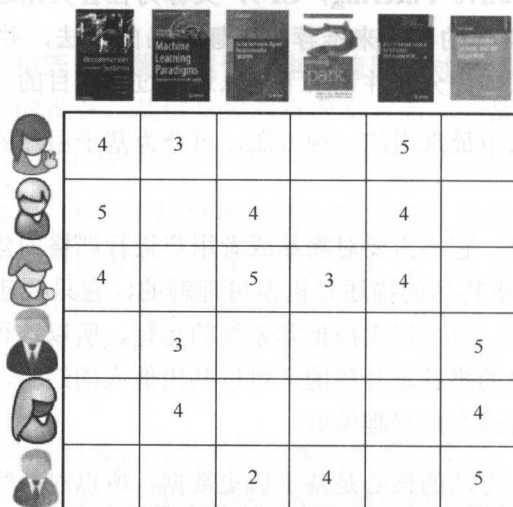
以上介绍的方法是推荐领域最常见的几种方法。但是可以看出，每种方法都不是完美的，因此实际应用中大都采用组合推荐算法，即把多种方法结合起来使用，各取所长，实现优势互补。

11.2 协同过滤推荐算法分析

由于协同过滤推荐算法的应用最为广泛，因此，本节深入研究该算法的原理。前面已经指出，协同过滤推荐分为基于用户的协同过滤和基于物品的协同过滤，下面分别予以介绍。

11.2.1 基于用户的协同过滤推荐

假设有一组用户，他们通过评分表现出对一组图书的喜好，用户对一本图书的喜好程度越高，就会给其越高的评分，范围是 1~5。我们可以用一个矩阵来表示这种用户与物品的评价关系（用户评分矩阵），如图 11-3 所示。行代表用户，列代表图书，矩阵的元素表示评分。例如，第一个用户（行 1）对第一本图书（列 1）的评分是 4，对第二本图书的评价是 3 分，等等。空的单元格代表用户未给图书评价。









	4	3			5	
	5		4		4	
	4		5	3	4	
		3				5
		4				4
			2	4		5

图 11-3 用户与物品的评价关系矩阵

采用基于用户的协同过滤，关键是要从原始的用户物品评价关系矩阵计算出用户之间的相似度矩阵。

为了说明方便，我们把用户记为 User1、User2、User3 等，图书记为 Item1、Item2、Item3 等。请仔细观察图 11-3 给出的关系矩阵。不难发现，User1 对 Item1、Item2、Item5 给出了评价，User2 对 Item1、Item3、Item5 给出了评价，他们都对 Item1 和 Item5 有兴趣，有两本共同的图书，所以，可以认为 User1 和 User2 有比较大的相似度。同理，我们发现 User2 与 User3 之间的相似度更高，因为他们都对第 1 本、第 3 本和第 5 本图书感兴趣，不同点只有第 4 本书。显然，User1 与 User4、User5 的相似度低则一些，因为只有一本共同书籍，而与最后一名用户完全不相似，因为他们之间没有一本共同书籍。

数学上，常见的做法是把用户的打分看成一个代表其特征的向量，例如 User1 的特征向量是 (4,3,0,0,5,0)，User2 对应的向量是 (5,0,4,0,4,0)，等等。于是，我们只要计算向量之间的相似度，就可以得到用户之间的相似度。这种做法非常直接。

有很多计算向量之间相似度的方法，本例使用了余弦相似性 (Cosine Similarity) 计算方法，如式 (11-1) 所示。当然，还有其他一些计算相似度的公式，读者可以参考相关文献。

$$CS(X,Y)=\frac{\sum x_i y_i}{\sqrt{\sum x_i^2 \times \sum y_i^2}} \quad (11-1)$$

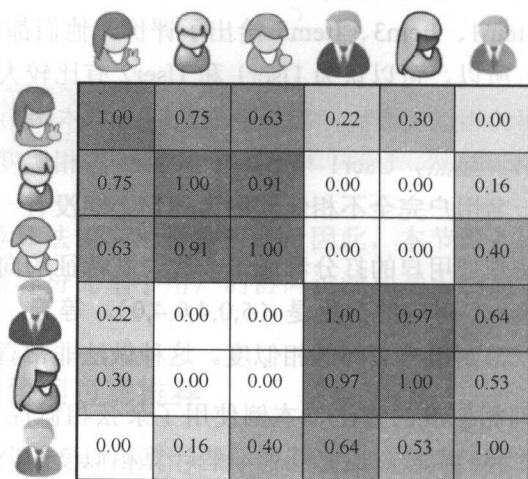
我们把 (4,3,0,0,5,0) 和 (5,0,4,0,4,0) 代入式 (11-1)，即可得到 0.75，此即 User1 与 User2 之间的相似性度量。

当计算出所有用户之间的相似性后，就能够得到一个用户相似度矩阵，如图 11-4 所示。这是一个对称矩阵，这意味着对它进行数学计算会有一些有用的特性。为了便于观察，单元格的背景颜色表明用户相似度的高低，更深的颜色表示他们之间更相似。

现在，我们已经为采用基于用户的协同过滤方法准备好了数据，接下来就可为用户生成推荐了。在一般情况下，对于一个给定的用户，这意味着找到最相似的用户，并推荐这些类似用户欣赏的物品，其中还需要根据用户相似度对这些物品进行加权处理。

继续研究这个示例。来看 User1，我们为其生成一些推荐。首先，找到与第一个用户最相似的另一些用户，这需要设定一个阈值 (Threshold)，规定相似度大于该阈值才被认定为有效，例如确定 0.60 是阈值，这样就能够为 User1 找到两名最相似的用户 User2 和 User3，即 $N=2$ 。然后，需要删除 User1 已经评价过的书籍，再给最相似用户正在阅读的

书籍加权，最后计算出推荐打分。由于 User1 已经评价了第 1、第 2 和第 5 本图书，所以候选的推荐图书是第 3 本、第 4 本和第 6 本。其中，第 3 本的推荐值可以这样计算： $(0.75 \times 4 + 0.63 \times 5) / (0.75 + 0.63)$ ，结果是 4.5 分。同理，对第 4 本图书的推荐值是 3 分，对第 6 本图书的推荐值是 0 分。最后将这个结果排序后呈现给 User1 即完成了推荐。



	1.00	0.75	0.63	0.22	0.30	0.00
	0.75	1.00	0.91	0.00	0.00	0.16
	0.63	0.91	1.00	0.00	0.00	0.40
	0.22	0.00	0.00	1.00	0.97	0.64
	0.30	0.00	0.00	0.97	1.00	0.53
	0.00	0.16	0.40	0.64	0.53	1.00

图 11-4 用户间的相似矩阵

推荐值的计算表达式如式 (11-2) 所示。

$$Rcom_i^t = \frac{\sum (Sim_{ij} \times Rank_j^t)}{\sum Sim_{ij}}, \quad i \neq j \quad (11-2)$$

式中， $Rcom_i^t$ 表示给用户 $User_i$ 的推荐物品 t 的推荐值， Sim_{ij} 表示 $User_i$ 与 $User_j$ 之间的相似度， $Rank_j^t$ 是 $User_j$ 给物品 t 给出的评价。

基于用户的协同过滤推荐算法在用户数不大的情况下有一定效果，但是实际应用中用户数往往非常大，例如主要电子商务网站的用户数达到了上亿的数量级，这时候，基于用户的推荐算法将难以实用化（特别是不能实时推荐）。

11.2.2 基于物品的协同过滤推荐

与用户数相比，物品的数量会少很多，因此，工业界比较倾向于采用基于物品的协同过滤推荐。

同样的，我们仍然以用户与物品之间的评价关系矩阵为基础。类似于基于用户的协同过滤，在基于物品的协同过滤中，我们要做的第一件事也是计算相似矩阵。然而，这一次

我们要看的是物品与物品之间的相似性，而不是用户之间的相似性。在我们这里，要计算出一本书和其他书的相似性，可以将评价同一本图书的所有用户评价值看成这本图书的特征向量，然后比较它们之间的余弦相似性度量值。具体过程读者可以自己练习一下，这里为节省篇幅就不再重复计算了。

图 11-5 给出了所有图书之间相似性度量值的对称矩阵。同样，单元格背景颜色的深浅表示相似度的高低，颜色越深表明相似度越高。

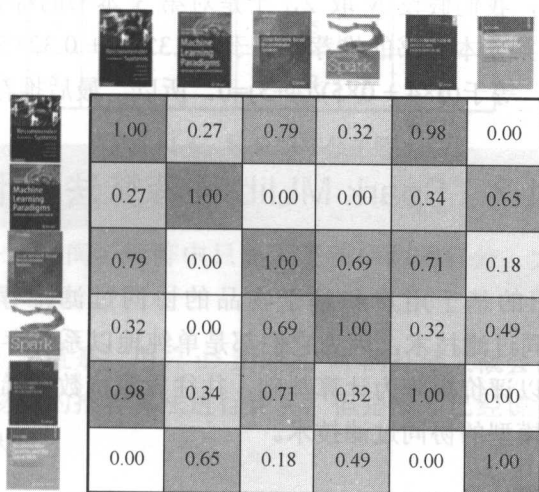


图 11-5 所有图书之间相似度的相似矩阵

知道了图书之间的相似度，就可以为用户进行推荐了。在基于物品的推荐方法中，向某用户推荐的物品，是该用户没有用过的其他最相似的物品。

现在让我们来为 User1 进行推荐。

在本例中，因为 User1 已经评价过第 1、第 2 和第 5 本图书，所以将被推荐第 3、第 4 和第 6 本图书。但是，需要给出一个推荐排序。由于 User1 对第 5 本书给出的评分最高 (5)，因此，我们可以简单比较一下第 3、第 4 和第 6 本图书与第 5 本书的相似度。很快看出，第 3 本图书与第 5 本图书的相似度为 0.71，第 4 本图书与第 5 本图书的相似度是 0.32，而第 6 本图书与第 5 本图书的相似度是 0，所以推荐第 3 和第 4 本。

同理，我们还可以再比较一下第 3、第 4 和第 6 本图书与第 1 本书的相似度，因为第 1 本图书是 User1 给出的第二高分的书。结果分别是 0.79、0.32 和 0，所以还是推荐第 3 和第 4 本图书给 User1。

实际上，我们需要综合考虑第 3、第 4 和第 6 本图书与第 1、5 本图书的比较结果。

显然，我们可以仿照公式 11-2，得到如下的综合评价表达式。

$$Rcom_t^i = \frac{\sum (Sim_t^{topN} \times Rank_{topN}^i)}{\sum Rank_{topN}^i}, i \neq j \quad (11-3)$$

式中， $Rcom_t^i$ 表示给用户 $User_i$ 推荐物品 t 的推荐值， Sim_t 表物品 t 与 $User_i$ 评价过的最高评分的前 N 名物品之间的相似度， $Rank_{topN}$ 是 $User_i$ 给出的对物品评分的前 N 名。

这样依据式 (11-3)，我们假设 N 取 2，于是对第 3 本书的综合推荐值等于 $(0.79 \times 4 + 0.71 \times 5) / (4 + 5) = 0.75$ ；对第 4 本图书的推荐值等于 $(0.32 \times 4 + 0.32 \times 5) / (4 + 5) = 0.32$ ；进一步计算第 6 本书的推荐值，等于 $(0 \times 4 + 0 \times 5) / (4 + 5) = 0$ 。所以，最后推荐第 3、4 本图书。

11.3 Spark MLlib 推荐算法应用

实际上，前面介绍的基于用户和基于物品的协同过滤推荐算法又称为基于记忆 (Memory Based) 的协同过滤技术，因为它们都是单纯地以系统存储的用户评分矩阵为推荐基础的。然而，仅仅以评价矩阵为计算基础，往往导致抗数据稀疏的能力较差，因此研究人员又发展出了基于模型的协同过滤技术。

11.3.1 ALS 算法原理

通常，用户对产品的评分矩阵是庞大而稀疏的，因此在非常稀疏的数据集上采用简单的用户 (或物品) 相似度比较进行推荐，直观上给人的感觉是这样做缺少依据。理论上分析一下我们也能理解，基于记忆的协同过滤方法实际上并没有充分挖掘数据集中的潜在因素。

本节介绍的 Spark MLlib 推荐方法交替最小二乘法 (Alternating Least Squares, ALS)，是一种基于模型的协同过滤推荐算法，其核心思想就是要进一步挖掘通过观察得到的所有用户给产品的打分，并通过引入用户特征矩阵 (User Features Matrix) 和物品特征矩阵 (Item Features Matrix)，来建立一个机器学习模型，然后利用采集的数据对这个模型进行训练 (反复迭代)，最后得到用于推荐计算的用户特征矩阵和物品特征矩阵，从而来推断 (也就是预测) 每个用户的喜好并向用户推荐适合的物品。

ALS 学习算法解决了用户与物品评价关系矩阵中的缺失因子问题，实现了用预测得到的缺失因子进行推荐。

考虑下面这个反映用户偏好的打分矩阵 (Rating Matrix)，如图 11-6 所示。

					4		8	
		6		1		7		
	4		3					5
		5	2					3
			?	7		1		
9					5			
7					3	5		

图 11-6 稀疏的打分矩阵

这个矩阵的每一行代表一个用户 (u_1, u_2, \dots, u_7), 每一列代表一个产品 (v_1, v_2, \dots, v_9)。用户的打分在 1~9 之间。矩阵中只显示了观察到的打分, 大部分元素都是缺失的(稀疏性)。

举例来说, 我们的问题就是, 用户 u_5 给产品 v_4 的打分大概会是多少? 我们当然可以按照传统的基于用户或物品的推荐算法进行计算, 但是刚才已经说了, 这样做效果并不理想, 而且计算量也非常大。

那么, ALS 的思路不同了。ALS 基于下面这个假设: 打分矩阵是近似低秩 (Low-Rank) 的; 换句话说, 打分矩阵 $A(m \times n)$ 可以用两个小矩阵 $U(m \times k)$ 和 $V(n \times k)$ 的乘积来近似表示, 即有:

$$A \approx U(m, k)V(n, k)^T \quad (11-4)$$

式中, k 远小于 m 和 n 。这样就把整个系统的自由度从 $O(mn)$, 降到了 $O((m+n)k)$ 。其实, ALS 的低秩假设是建立在客观存在的合理性基础上的。例如, 用户特征当然有很多很多, 如年龄、性别、职业、身高、学历、婚姻、地区、存款等, 可以说不胜枚举, 然而, 我们没有必要把用户的所有特征统统都用起来, 因为并不是所有特征都起同样的作用。例如, 我们后面展示的电影推荐示例中, 用户特征矩阵仅仅包含了用户编号、性别、年龄、职业、邮编这 5 个字段。同样, 物品的属性也有很多, 以电影为例, 可以有主演、导演、特效、剧情、类型, 等等。但实际应用中我们只需要描述少数关键属性即可。我们后面的举例中, 仅仅考虑了三个属性, 即电影编号、电影名和电影类别(当然这只是举例, 到底 k 取什么值, 可以采用系统自适应调节方法, 通过应用逐步找到最佳 k 值)。

总之, ALS 算法的巧妙之处就在于, 引入了两个特征矩阵, 一个是用户特征矩阵, 用 U 表示, 另一个是物品特征矩阵, 用 V 表示。这两个矩阵都比较小。

接下来的问题是怎样得到这两个抽象的低维空间。既然已经假设打分矩阵 A 可以通过 UV^T 来近似, 那么一个最直接的可以量化的东西就是通过 U 和 V 重构 A 时产生的误差。在 ALS 里, 使用式 (11-5) 给出的 Frobenius 范数 (又称为 Euclid 范数):

$$\|A - UV^T\|_F^2 \quad (11-5)$$

来表示重构误差, 也就是每个元素的重构误差的平方和, 如式 (11-6) 所示。

$$\sum_{i,j \in R} (a_{ij} - u_i^T v_j)^2 \quad (11-6)$$

这里存在一个问题, 由于只观察到部分的打分, A 中有大量未知元是需要推断的, 所以这个重构误差包含了未知数。解决方案很简单, 就是只计算对已知打分的重构误差。当然, 也可以先用一个简单的方法把评分矩阵填满, 再进行重构误差计算, 但是这样做似乎也没有太多道理。

总之, ALS 算法就是求解下面的优化问题:

$$\text{Arg min} \sum_{i,j \in R} (a_{ij} - u_i^T v_j)^2 \quad (11-7)$$

经过上面的处理, 一个协同推荐问题通过低秩假设被成功地转换成了一个优化问题。但是, 这个优化问题怎么解? 不要忘记, 我们的目标是求出 U 和 V 这两个矩阵。

答案就在 ALS 的名字里, 即交替最小二乘。由于 ALS 的目标函数不是凸的, 而且变量互相耦合在一起, 所以它并不容易求解。但如果把用户特征矩阵 U 和产品特征矩阵 V 固定其一, 这个问题立刻就变成了一个凸的而且是可拆分的问题了。例如, 固定 U 求 V , 这个问题是经典的最小二乘问题。所谓交替, 就是指先随机生成 $U(0)$, 然后固定它, 去求解 $V(0)$; 再固定 $V(0)$, 然后求解 $U(1)$, 这样交替进行下去。因为每一次迭代都会降低重构误差, 并且误差是有下界的, 所以 ALS 一定会收敛。但由于问题是非凸的, 所以 ALS 并不保证会收敛到全局最优解。然而在实际应用中, ALS 对初始点不是很敏感, 且是不是全局最优解也不会有大的影响。

ALS 算法可以大体描述如下。

第一步, 用小于 1 的数随机初始化 V 。

第二步, 在训练数据集上反复迭代, 交替计算 U 和 V , 直到 RMSE (均方根误差,

一种常用的离散性度量方法) 值收敛或迭代次数足够多。

第三步, 返回 UV^T , 进行预测推荐。

之所以说上述算法是一个大体描述, 是因为第二步里面还包含有如何计算 U 和 V 的表达式, 它们是通过求偏导推出的。

11.3.2 ALS 的应用设计

本节给出的实例使用 ALS 算法对 GroupLens Research (<http://grouplens.org/datasets/movielens/>) 提供的数据进行学习并推荐。该数据为一组从 20 世纪 90 年末到 21 世纪初由 MovieLens 用户提供的电影评价数据, 包括评分、电影元数据 (风格类型和年代), 以及关于用户的人口统计学数据 (年龄、邮编、性别和职业等)。根据不同需求, 该组织提供了不同大小的样本数据, 不同样本信息中包含三种数据: 评分、用户信息和电影信息。下面我们首先来看看待处理的数据是什么样子, 然后给出应用程序并进行分析。

1. 输入数据

(1) 评分文件 (rating.dat)。该评分数据有四个字段, 格式为 UserID::MovieID::Rating::Timestamp, 分别为用户编号、电影编号、评分、评分时间戳。其中各个字段说明如下:

- 用户编号范围 1~6040。
- 电影编号 1~3952。
- 电影评分为五星评分, 范围 0~5。
- 评分时间戳单位是秒。
- 每个用户至少有 20 个电影评分。

ratings.dat 文件中的数据样本如下所示。

```
1::1193::5::978300760
1::661::3::978302109
1::914::3::978301968
1::3408::4::978300275
1::2355::5::978824291
1::1197::3::978302268
1::1287::5::978302039
1::2804::5::978300719
.....
```

(2) 用户信息文件 (users.dat)。分为五个字段, 格式为 UserID::Gender::Age::Occupation::Zip-code, 分为为用户编号、性别、年龄、职业、邮编。各个字段说明如下。

- 用户编号范围 1~6040。
- 性别, 其中 M 为男性, F 为女性。
- 年龄范围, 不同的数字代表不同的年龄段, 如 25 代表 25~34 岁。
- 职业信息, 在测试数据中提供了 21 中职业分类。
- 地区邮编。

users.dat 文件的数据样本如下所示:

```
1::F::1::10::48067
2::M::56::16::70072
3::M::25::15::55117
4::M::45::7::02460
5::M::25::20::55455
6::F::50::9::55117
7::M::35::1::06810
8::M::25::12::11413
.....
```

(3) 电影信息 (movies.dat)。电影数据分为三个字段, 格式为 MovieID::Title::Genres, 分为为电影编号、电影名、电影类别, 其中各个字段说明如下。

- 电影编号 1~3952。
- 由 IMDB 提供电影名称, 其中包括电影上映年份。
- 电影分类, 这里使用实际分类名而非编号, 如 Action、Crime 等。

movies.dat 的数据样本如下所示。

```
1::Toy Story (1995)::Animation|Children's|Comedy
2::Jumanji (1995)::Adventure|Children's|Fantasy
3::Grumpier Old Men (1995)::Comedy|Romance
4::Waiting to Exhale (1995)::Comedy|Drama
5::Father of the Bride Part II (1995)::Comedy
6::Heat (1995)::Action|Crime|Thriller
7::Sabrina (1995)::Comedy|Romance
8::Tom and Huck (1995)::Adventure|Children's
.....
```

2. 程序分析

下文给出了完整的 ALS 推荐算法应用程序，并通过在程序中添加注释语句，来说明程序各个部分的功能。读者也可以结合 Scala 语法进行更加细致的理解。

Spark MLlib 中 ALS 算法的实现有如下参数：numBlocks 是用于并行化计算的分块个数（设置为-1 时为自动配置）；rank 是模型中隐性因子的个数；iterations 是迭代的次数；lambda 是 ALS 的正则化参数；implicitPrefs 决定了是用显性反馈 ALS 的版本还是用隐性反馈数据集的版本；alpha 是一个针对于隐性反馈 ALS 版本的参数，这个参数决定了偏好行为强度的基准，但本例实际上没有应用 alpha 参数。

```
package com.csu
import java.util.Random
import org.apache.spark.{SparkConf, SparkContext}
import org.apache.spark.mllib.recommendation.{ALS,
                                                MatrixFactorizationModel, Rating}
import org.apache.spark.rdd._object moviesALS
{
  def main(args: Array[String])
  {
    if (args.length != 1)
    {
      println("Please input movieLens Home directory,
              e.g: /tmp/data/")
      System.exit(1)
    }
    val movieLensHomeDir = args(0)
    if(!new java.io.File(movieLensHomeDir + "/ratings.dat").exists)
    {
      println("File rating.dat is not exist under directory:"
              + movieLensHomeDir)
      System.exit(1)
    }
    if(!new java.io.File(movieLensHomeDir + "/movies.dat").exists)
    {
      println("File movies.dat is not exist under directory:"
              + movieLensHomeDir)
      System.exit(1)
    }
  }
  // 建立 Spark 环境
```



```

val conf = new SparkConf()
    .setMaster("local")
    .setAppName("moviesALS")
val sc = new SparkContext(conf)
// 加载评分文件 ratings.dat 和电影文件 movies.dat
val ratings = sc.textFile(movieLensHomeDir + "/ratings.dat").map
{
    line =>
    val fields = line.split(":")
    // format: (timestamp % 10, Rating(userId, movieId, rating))
    (fields(3).toLong % 10, Rating(fields(0).toInt,
        fields(1).toInt, fields(2).toDouble))
}
val movies = sc.textFile(movieLensHomeDir + "/movies.dat").map
{
    line =>
    val fields = line.split(":")
    // format: (movieId, movieName)
    (fields(0).toInt, fields(1))
}.collect.toMap
val numRatings = ratings.count
val numUsers = ratings.map(_._2.user).distinct.count
val numMovies = ratings.map(_._2.product).distinct.count
println("Got " + numRatings + " ratings from "
    + numUsers + " users on " + numMovies + " movies.")
// 提取一个得到最多评分的电影子集，以便进行评分启发
val mostRatedMovieIds = ratings.map(_._2.product) // extract movie id
    .countByValue // count ratings per movie
    .toSeq // convert map to Seq
    .sortBy(_._2) // sort by rating count
    .take(50) // take 50 most rated
    .map(_._1) // get their ids
val random = new Random(0)
val selectedMovies = mostRatedMovieIds.filter(
    x => random.nextDouble() < 0.2)
    .map(x => (x, movies(x)))
    .toSeq
// 引导或启发评分
val myRatings = elicitateRatings(selectedMovies)
val myRatingsRDD = sc.parallelize(myRatings, 1)

```

```

// 将评分文件分成训练集 (60%), 验证集 (20%)和测试集 (20%)
// 并且缓存这些数据
val numPartitions = 20
val training = ratings.filter(x => x._1 < 6)
    .values
    .union(myRatingsRDD)
    .repartition(numPartitions)
    .persist
val validation = ratings.filter(x => x._1 >= 6 && x._1 < 8)
    .values
    .repartition(numPartitions)
    .persist
val test = ratings.filter(x => x._1 >= 8).values.persist
val numTraining = training.count
val numValidation = validation.count
val numTest = test.count
println("Training: " + numTraining + ", validation: " +
    numValidation + ", test: " + numTest)

// 训练模型, 并在验证集上评估模型
val ranks = List(8, 12)
val lambdas = List(0.1, 11.0)
val numIters = List(10, 20)
var bestModel: Option[MatrixFactorizationModel] = None
var bestValidationRmse = Double.MaxValue
var bestRank = 0
var bestLambda = -1.0
var bestNumIter = -1
for (rank <- ranks; lambda <- lambdas; numIter <- numIters)
{
    val model = ALS.train(training, rank, numIter, lambda)
    val validationRmse = computeRmse(model, validation,
        numValidation)
    println("RMSE (validation) = " + validationRmse + "
        for the model trained with rank = "
        + rank + ", lambda = " + lambda + ",
        and numIter = " + numIter + ".")
    if (validationRmse < bestValidationRmse)
    {
        bestModel = Some(model)
        bestValidationRmse = validationRmse
        bestRank = rank
    }
}

```

```

        bestLambda = lambda
        bestNumIter = numIter
    }
}

// 在测试集上评估得到的最佳模型
val testRmse = computeRmse(bestModel.get, test, numTest)
println("The best model was trained with rank = " + bestRank + "
        and lambda = " + bestLambda + ", and numIter = "
        + bestNumIter + ", and its RMSE on the test set is "
        + testRmse + ".")

// 设置朴素基线并与最佳模型比较
val meanRating = training.union(validation).map(_._rating).mean
val baselineRmse = math.sqrt(test.map(x => (meanRating - x.rating)
        * (meanRating - x.rating)).reduce(_ + _) / numTest)
val improvement = (baselineRmse - testRmse) / baselineRmse * 100
println("The best model improves the baseline by "
        + "%1.2f".format(improvement) + "%.")

// 产生个性化推荐
val myRatedMovieIds = myRatings.map(_._product).toSet
val candidates = sc.parallelize(movies.keys.filter
        (!myRatedMovieIds.contains(_))).toSeq
val recommendations = bestModel.get
        .predict(candidates.map((0, _)))
        .collect
        .sortBy(- _._rating)
        .take(50)
var i = 1
println("Movies recommended for you:")
recommendations.foreach { r =>
    println("%2d".format(i) + ": " + movies(r._product))
    i += 1
}
// clean up
sc.stop()
}

//计算 RMSE (Root Mean Squared Error)
def computeRmse(model: MatrixFactorizationModel, data: RDD[Rating], n: Long) = {
    val predictions: RDD[Rating] = model.predict(data.map(x => (x.user,
        x.product)))
    val predictionsAndRatings = predictions.map(x => ((x.user,
        x.product), x.rating))

```

```

.join(data.map(x => ((x.user, x.product), x.rating)))
.values
math.sqrt(predictionsAndRatings.map(x => (x._1 - x._2) *
(x._1 - x._2)).reduce(_ + _) / n)
}
// 从命令行获取引导性评分
def elicitateRatings(movies: Seq[(Int, String)]) = {
  val prompt = "Please rate the following movie (1-5 (best),
    or 0 if not seen):"

  println(prompt)
  val ratings = movies.flatMap { x =>
    var rating: Option[Rating] = None
    var valid = false
    while (!valid) {
      print(x._2 + ": ")
      try {
        val r = Console.readInt
        if (r < 0 || r > 5) {
          println(prompt)
        } else {
          valid = true
          if (r > 0) {
            rating = Some(Rating(0, x._1, r))
          }
        }
      } catch {
        case e: Exception => println(prompt)
      }
    }
    rating match {
      case Some(r) => Iterator(r)
      case None => Iterator.empty
    }
  }
  if(ratings.isEmpty) {
    error("No rating provided!")
  } else {
    ratings
  }
}

```

3. 运行程序

首先启动 IDEA，如图 11-7 所示。

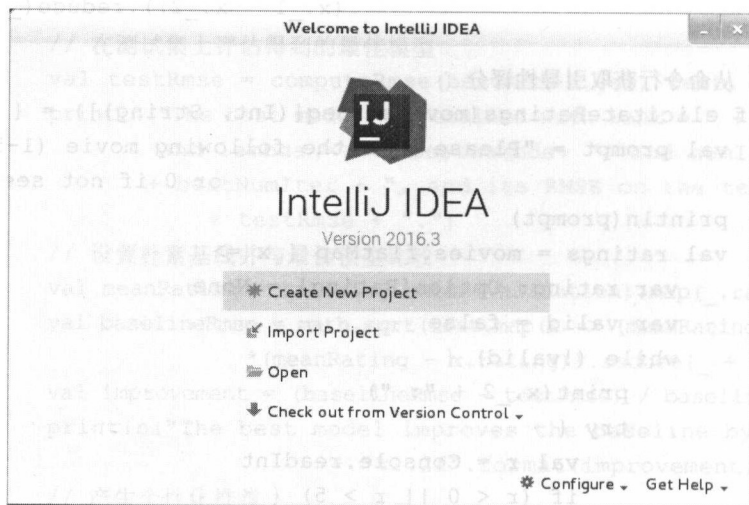


图 11-7 启动 IDEA

单击“Create New Project”，创建一个 Scala 新工程，如图 11-8 所示。

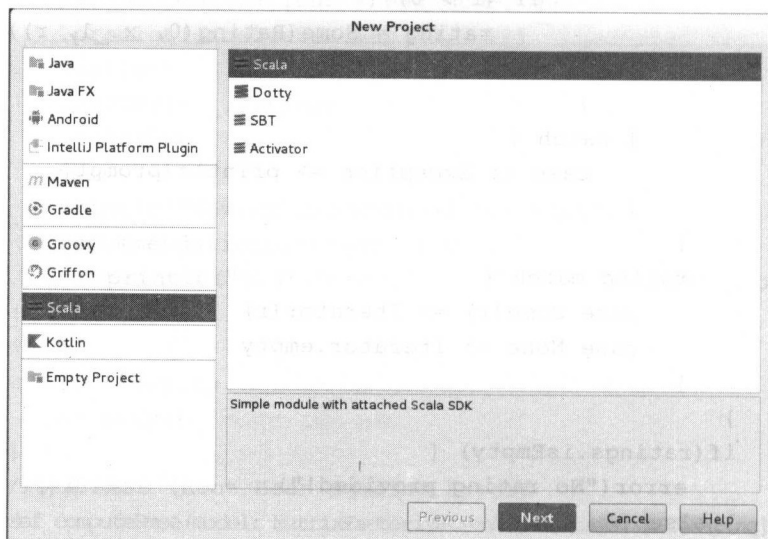


图 11-8 创建一个 Scala 新工程

在图 11-8 中，选择 Scala 后单击“Next”按钮进入工程基本设置页面，如图 11-9 所示。

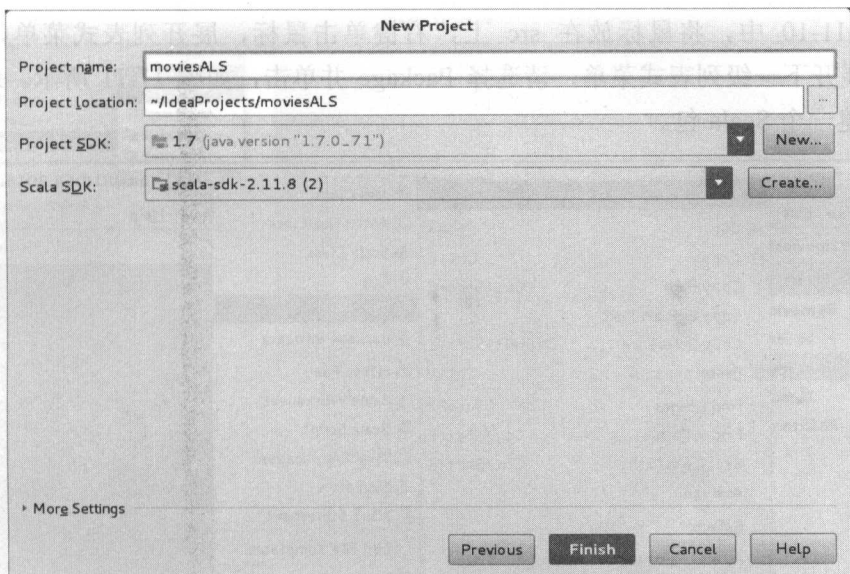


图 11-9 工程名称与位置等基本设置

在图 11-9 中，开发人员需要设置 Project name、Project location 和 Project SDK 等。这里我们给工程取名为 moviesALS（读者可以任意取一个名称），工程位置（Project location）自动设置，而 Project SDK 与 Scala SDK 是前面安装 IDEA 时已经设置好的，所以会自动显示出来。单击“Finish”按钮后，进入如图 11-10 所示的画面。

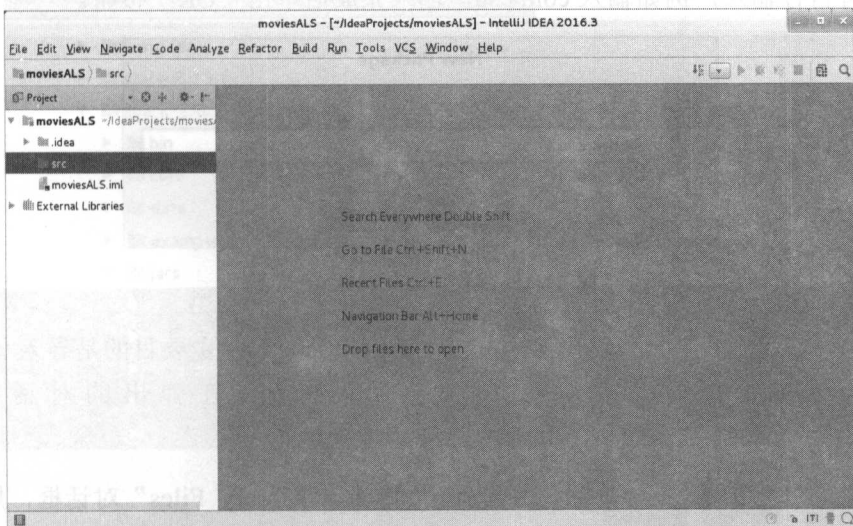


图 11-10 单击图 11-9 中“Finish”按钮后进入开发环境

在图 11-10 中，将鼠标放在 `src` 上，右键单击鼠标，展开列表式菜单，然后选择 `New`，再展开下一级列表式菜单，请选择 `Package` 并单击，如图 11-11 所示。这意味着我们准备创建一个 `Scala` 包。

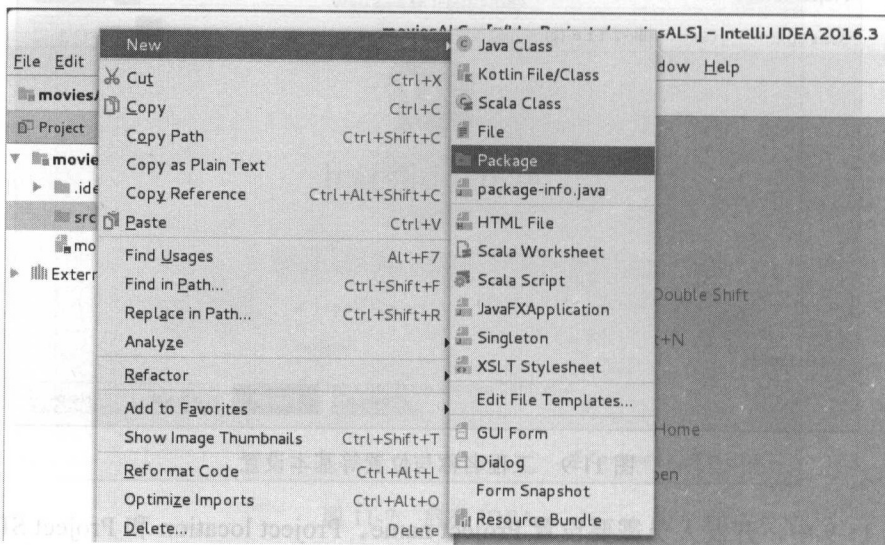


图 11-11 创建一个 `Scala` 包

选择 `Package` 后，系统弹出如图 11-12 所示的对话框，要求用户输入新的 `Package` 名称，用户可自行输入，例如输入 `com.csu`，输入完成后单击“OK”按钮。

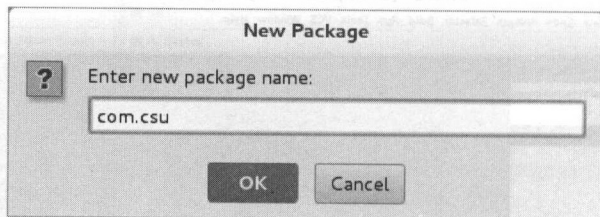


图 11-12 输入新的 `Package` 名称

接下来，我们先配置一下工程结构（`Project Structure`），主要目的是导入 `Spark` 依赖包。依次选择主界面的“`File→Project Structure`”，在弹出的对话框中选择“`Libraries→+→Java`”，如图 11-13 所示。

单击图 11-13 中的“`Java`”后，系统弹出“`Select Library Files`”对话框，用户可以开始导入安装好的 `Spark` 目录（例如 `spark-2.0.1-bin-hadoop2.6`）下 `jars` 内的所有 `jar` 包，如图 11-14 所示。

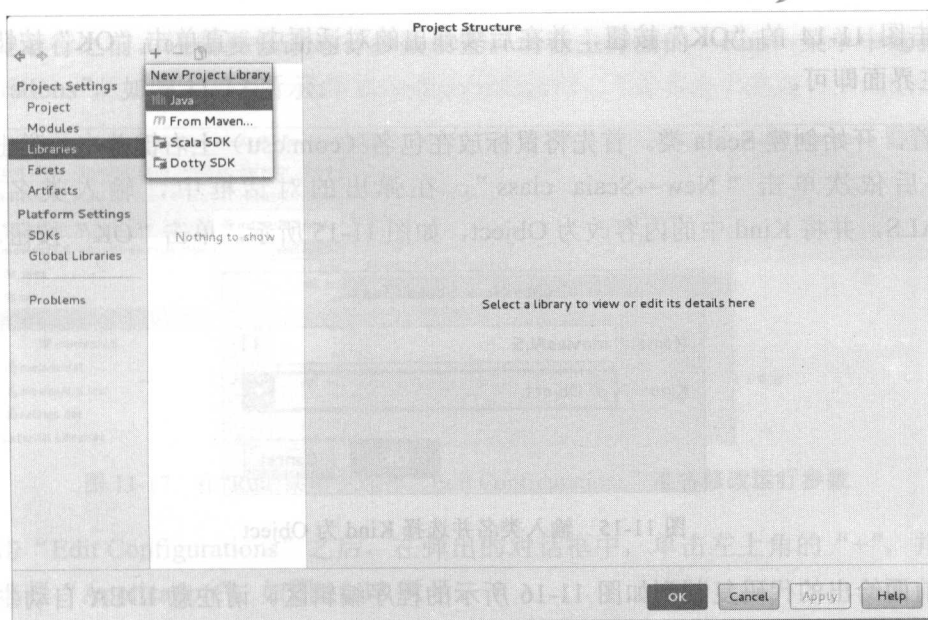


图 11-13 在 Project Structure 中选择“Libraries→++→Java”

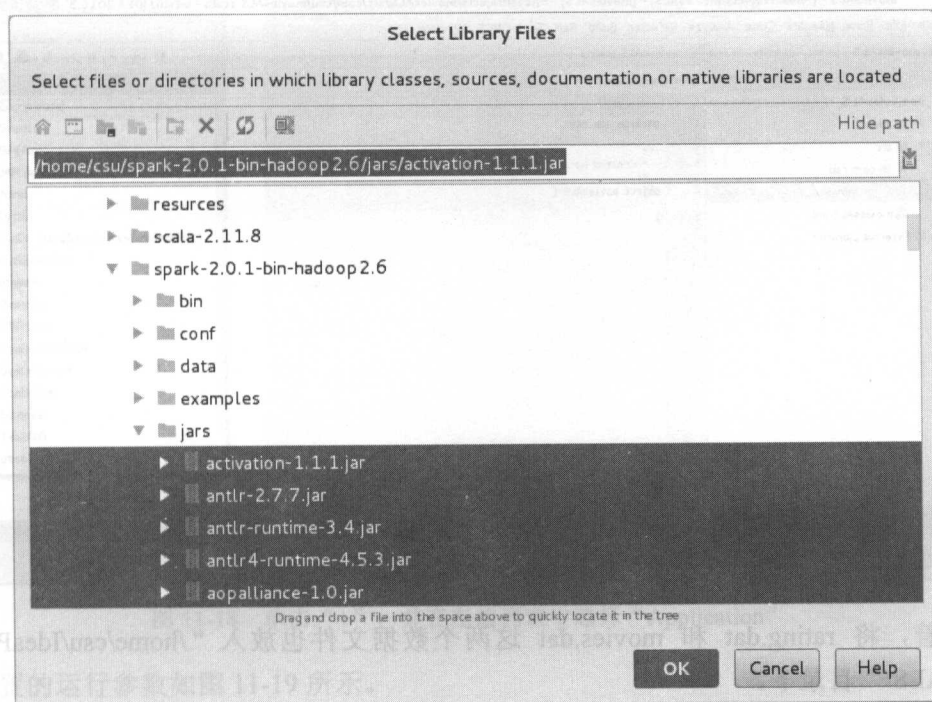


图 11-14 导入 Spark jars 目录下的所有 jar 包

单击图 11-14 的“OK”按钮，并在后续弹出的对话框中一直单击“OK”按钮，回到 IDEA 主界面即可。

接着，开始创建 Scala 类。首先将鼠标放在包名（com.csu）上右键单击，弹出列表菜单，然后依次单击“New→Scala class”，在弹出的对话框中，输入类名称，如 moviesALS，并将 Kind 中的内容改为 Object，如图 11-15 所示，单击“OK”按钮。

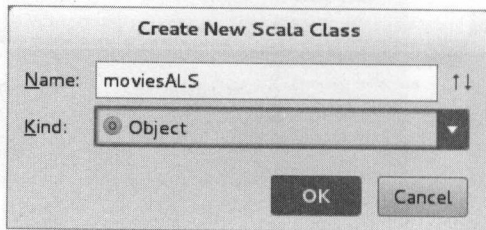


图 11-15 输入类名并选择 Kind 为 Object

将前面给出的代码复制到如图 11-16 所示的程序编辑区。请注意 IDEA 自动生成的部分代码，如 package com.csu 等，读者复制代码时不要重复。

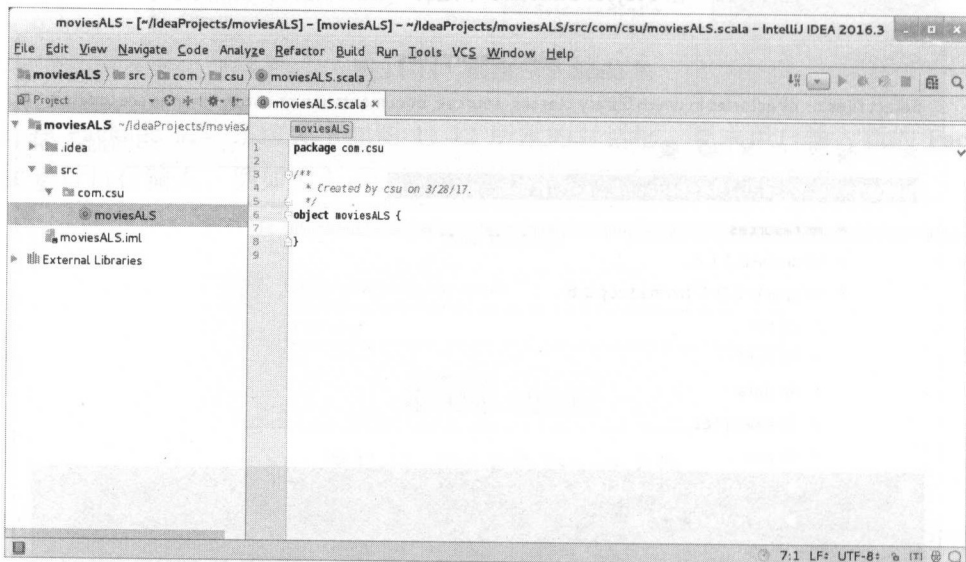


图 11-16 将代码贴入编辑区

接着，将 rating.dat 和 movies.dat 这两个数据文件也放入“/home/csu/IdeaProjects/moviesALS/”目录下。

如果顺利，编辑区内应当没有错误，这时候可以准备运行程序了。

但是运行之前，还需要修改运行参数设置。请在主界面的“Run”菜单中选择“Edit Configurations”，如图 11-17 所示。

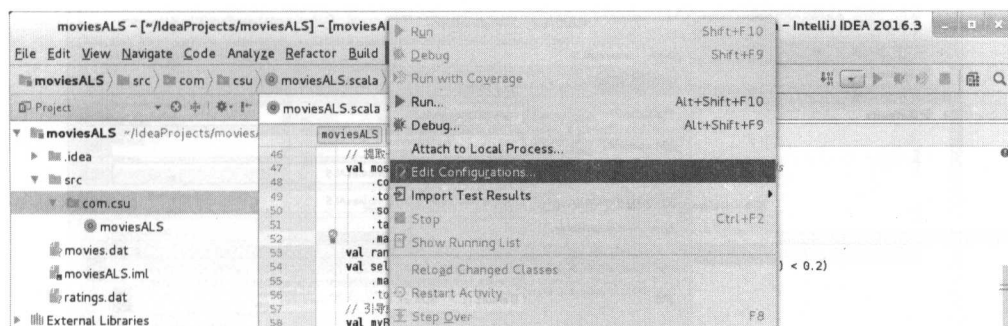


图 11-17 在“Run”菜单中选择“Edit Configurations”准备修改运行参数

单击“Edit Configurations”之后，在弹出的对话框中，单击左上角的“+”，并在左边列表中选择“Application”，如图 11-18 所示。

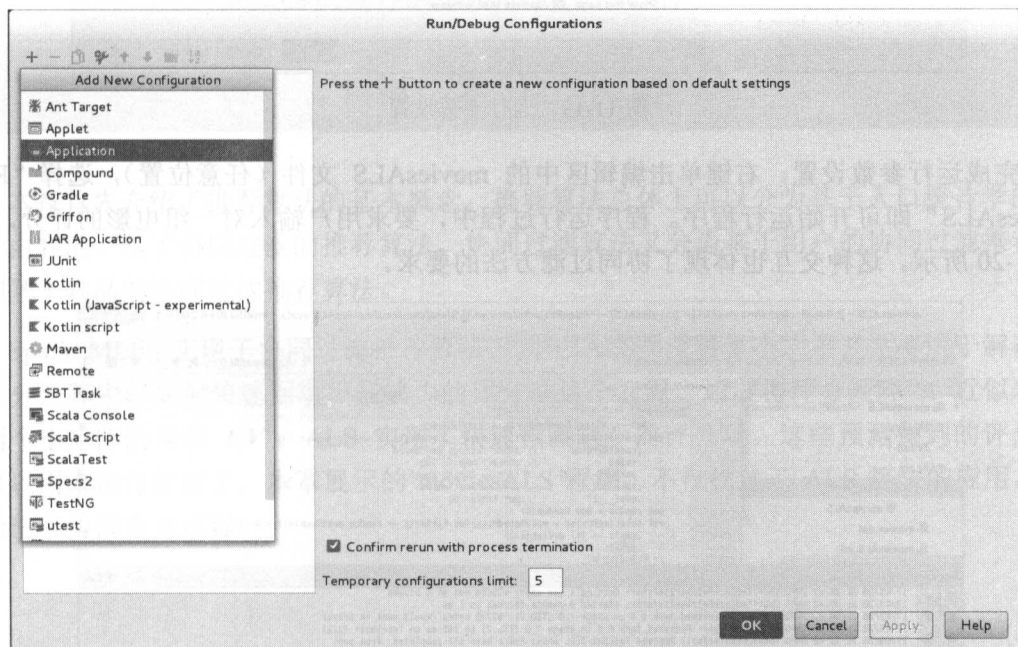


图 11-18 单击“+”，并在左边列表中选择“Application”

设置的运行参数如图 11-19 所示。

其中，Name 为 moviesALS，Main class 为“com.csu.moviesALS”，Program arguments

和 Working directory 均为事先创建好的“/home/csu/IdeaProjects/moviesALS”。请特别注意，待处理的数据文件放在什么目录下，Program arguments 的设置就要与那个目录一致。

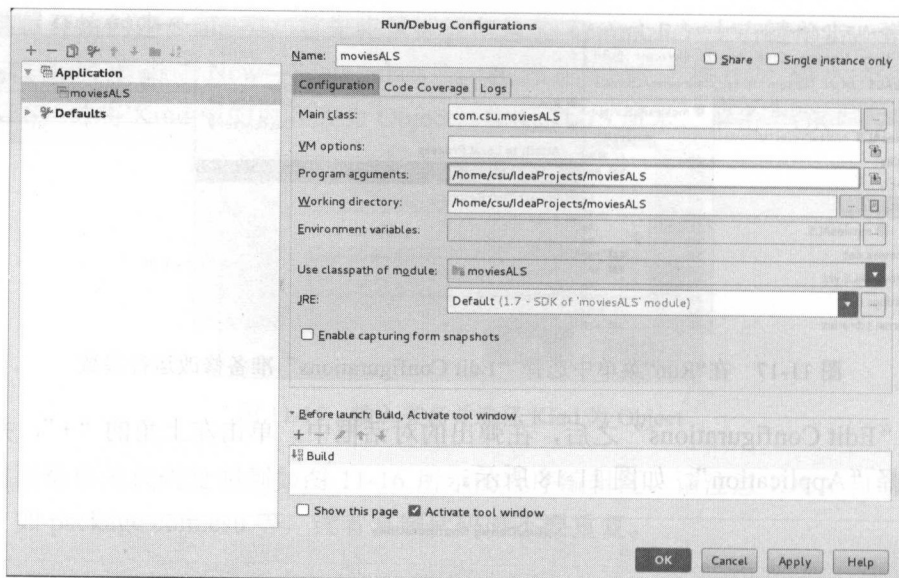


图 11-19 运行参数设置

完成运行参数设置，右键单击编辑区中的 moviesALS 文件（任意位置），选择“Run moviesALS”即可开始运行程序。程序运行过程中，要求用户输入对一组电影的评分，如图 11-20 所示。这种交互也体现了协同过滤方法的要求。

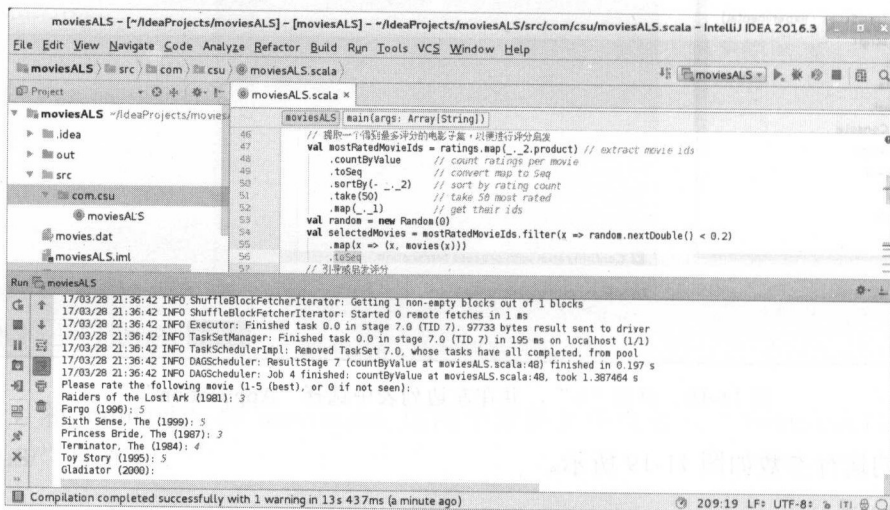


图 11-20 程序要求用户输入对一组电影的评分

图 11-21 给出了程序运行的最后结果，即程序在标准输出打印出一组（这里是 50 个）推荐的电影名称，并按照推荐的优先顺序依次排列。

```

moviesALS - [~/IdeaProjects/moviesALS] - [moviesALS] - ~/IdeaProjects/moviesALS/src/com/csu/moviesALS.scala - IntelliJ IDEA 2016.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
moviesALS src com csu moviesALS.scala
Project moviesALS
moviesALS main(args: Array[String])
// 选取一个评分最多评分的电影子集，以便进行评分预测
val sortedMovieIds = ratings.map(_._2.product) // extract movie id
    .countByValue // count ratings per movie
    .toSeq // convert map to Seq
    .sortBy(_._2) // sort by rating count
    .take(50) // take 50 most rated
    .map(_._1) // get their ids
val random = new Random(0)
val selectedMovies = sortedMovieIds.filter(x => random.nextDouble() < 0.2)
    .map(x => (x, movies(x)))
// 打印推荐评分
Run moviesALS
17/03/28 21:40:39 INFO TaskSchedulerImpl: Removed TaskSet 1660.0, whose tasks have all completed, from pool
17/03/28 21:40:39 INFO DAGScheduler: ResultStage 1660 (collect at moviesALS.scala:114) finished in 0.110 s
17/03/28 21:40:39 INFO DAGScheduler: Job 77 finished: collect at moviesALS.scala:114, took 0.256489 s
Movies recommended for you:
1: Specials, The (2000)
2: Titanic (1997)
3: Grimalt (1993)
4: 42 Up (1996)
5: Toy Story 2 (1999)
6: Leather Jacket Love Story (1997)
7: Braindead (1992)
8: Trick (1999)
9: Star Maps (1997)
10: Relax... It's Just Sex (1996)
11: Small Wonders (1996)
12: Chushingura (1962)
Compilation completed successfully with 1 warning in 13s 437ms (4 minutes ago)
10:45:11 LF: UTF-8

```

图 11-21 程序给出的一组推荐电影名称

11.4 本章小结

本章首先介绍了推荐算法的基本概念。推荐算法总体上可以分为基于人口统计学的、基于内容的、基于协同过滤的推荐算法。协同过滤算法又分为基于用户的协同过滤推荐算法和基于物品的协同过滤推荐算法。

Spark MLlib 实现了协同过滤推荐算法，其中 ALS（交替最小二乘）主要用于解决稀疏评分矩阵中，如何快速回填那些缺少的评分项这个问题。通过将评分矩阵 A 近似表示成两个小矩阵的乘积 UV^T ，ALS 实现了快速预测缺少的评分项。这些预测得到的评分项就可以用来进行推荐了。本章展示的 moviesALS 程序，不仅描述了 ALS 算法的应用，而且给出了协同交互过程。

本章通过一个完整的销售数据分析系统设计，展示如何利用 Hadoop 的各种组件，开发实际的大数据分析系统。本章运用到的组件包括 HDFS、MySQL、Eclipse、Phoenix、HBase 和 Tomcat，所展示的数据和应用均来自真实场景，对读者有较高的参考价值。

12.1 数据采集

我们知道，大数据应用开发的第一步是数据采集。走好这一步非常关键，但往往也是最困难的一步。本节主要介绍采用 WebCollector 开发的数据采集模块，需要应用 JDK、Eclipse、MySQL 数据库等组件。我们这里的数据采集采用 Windows 平台，这也是实际生产中的常用方式。

12.1.1 在 Windows 下安装 JDK

首先从 <http://www.Oracle.com> 下载 Windows 环境下的 JDK 安装包，也可以直接到本书的第 12 章软件资源文件夹中找到 jdk-8u77-windows-x64.exe。双击该文件即可开始进行安装，安装程序的首页如图 12-1 所示。

JDK 的缺省安装位置是 “C:\program files\java”。

上述安装实际上有三项内容，第一是开发工具，第二是源代码，第三是 JRE。JRE 是 Java 运行时环境，包含了 Java 虚拟机、Java 基础类库，是运行 Java 程序所需要的软件环境。JDK 即 Java 开发工具包，是程序员使用 Java 语言编写程序所需的开发工具包，是提供给程序员使用的。JDK 包含了 JRE，同时还包含了编译 Java 源码的编译器 javac，也包

含了很多 Java 程序调试和分析的工具，如 jconsole、jvisualvm 等工具软件，并提供 Java 程序编写所需的文档和 demo 例子程序。

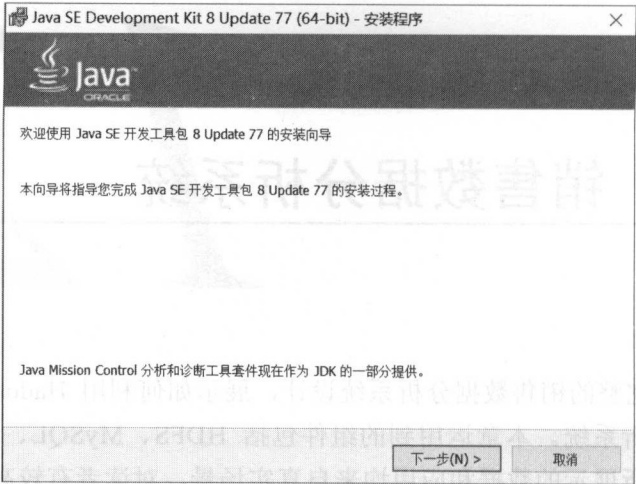


图 12-1 在 Windows 环境下开始安装 JDK

接下来需要配置环境变量。以 Windows10 系统为例，用鼠标右击“开始”图标，在弹出的列表中，选择“系统→高级系统设置→系统属性→高级→环境变量”，在弹出的“环境变量”对话框中，选择“系统变量”（该页下半部分）中的“Path”，如图 12-2 所示。

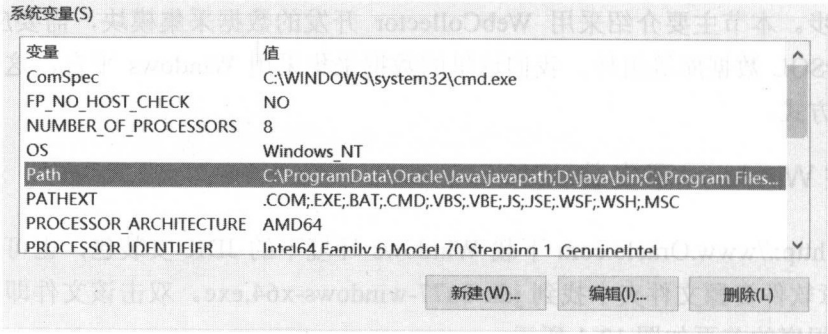


图 12-2 准备编辑环境变量

在图 12-2 中选择“Path”后，单击“编辑(I)…”按钮，弹出如图 12-3 所示的编辑对话框，在最下面的编辑框内输入“C:\program files\java\jdk1.8.0_77”。

读者注意，在图 12-3 中输入的具体内容，取决于自己计算机的安装情况，例如，如果 JDK 的安装目录是“D:\java\jdk1.8.0_77”，则输入的环境变量就是“D:\java\jdk1.8.0_77\bin”。输入完成后，单击“确定”按钮。

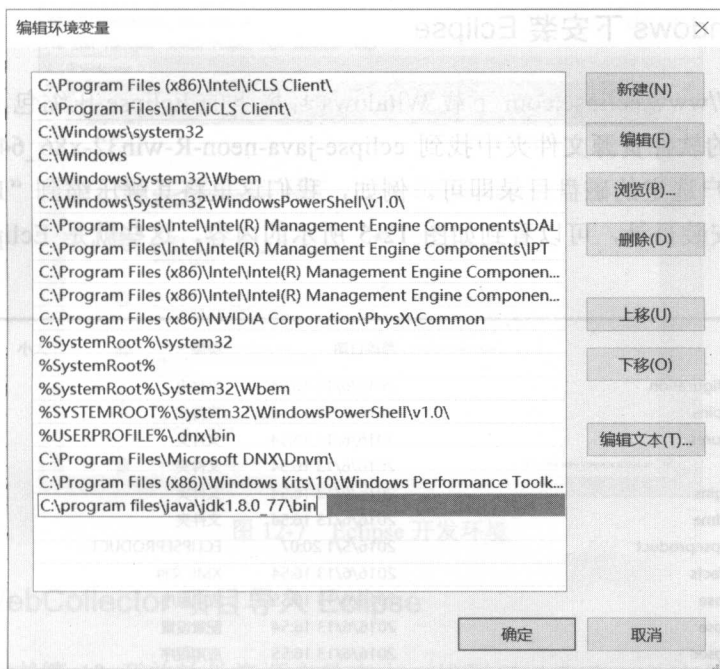


图 12-3 输入 JDK 环境变量

要验证环境变量是否配置成功，可以打开命令行窗口，输入 `javac`，如果返回的信息如图 12-4 所示，表明 Java 环境已经安装完成。



图 12-4 验证 Java 环境变量是否配置成功

12.1.2 在 Windows 下安装 Eclipse

首先从 <http://www.eclipse.com> 下载 Windows 环境下的 Eclipse 压缩包，读者也可以直接到本书第 12 章的软件资源文件夹中找到 eclipse-java-neon-R-win32-x86_64.zip。将该文件解压缩后放在用户选定的磁盘目录即可。例如，我们这里将其解压缩到“D:\Eclipse”。然后进入 Eclipse 安装目录，可以看到如图 12-5 所示的内容，这些就是 Eclipse 的文件和子目录。

名称	修改日期	类型	大小
configuration	2016/6/13 16:54	文件夹	
dropins	2016/6/13 16:54	文件夹	
features	2016/6/13 16:54	文件夹	
p2	2016/6/13 16:54	文件夹	
plugins	2016/6/13 16:54	文件夹	
readme	2016/6/13 16:54	文件夹	
.eclipseproduct	2016/5/1 20:07	ECLIPSEPRODUCT ...	1 KB
artifacts	2016/6/13 16:54	XML 文件	129 KB
eclipse	2016/6/13 16:55	应用程序	313 KB
eclipse	2016/6/13 16:54	配置设置	1 KB
eclipsec	2016/6/13 16:55	应用程序	25 KB

图 12-5 Eclipse 安装目录的内容

双击 eclipse.exe 文件即可启动 Eclipse，系统首先弹出“选择工作空间”对话框，可根据计算机实际情况选择设置。例如，直接在编辑框里输入“D:\eclipseworkspace”，系统将创建该目录，如图 12-6 所示。

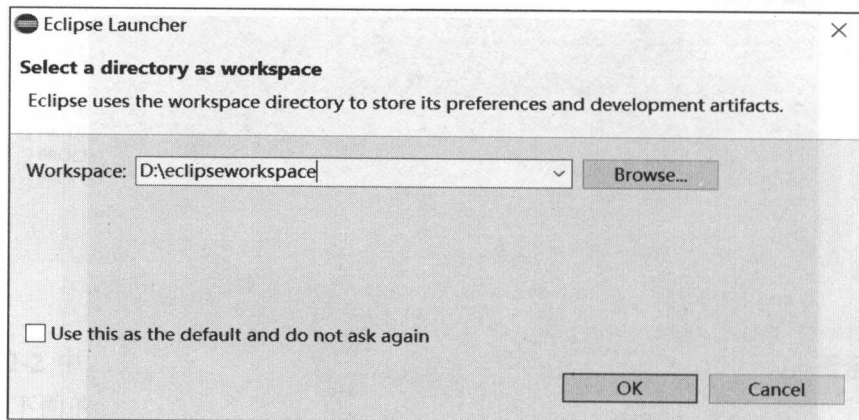


图 12-6 输入或选择一个目录作为工作空间 (workspace)

单击图 12-6 中的“OK”按钮后，进入 Eclipse 开发环境，如图 12-7 所示。

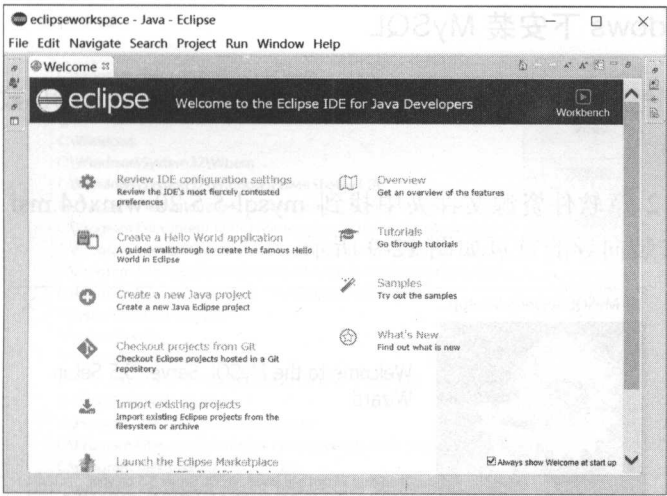


图 12-7 Eclipse 开发环境

12.1.3 将 WebCollector 项目导入 Eclipse

请读者到本书第 12 章的软件资源文件夹中，找到 WebCollector 文件夹，将其复制到上面创建的 eclipseworkspace 目录下。然后在图 12-7 所示的 Eclipse 主界面中选择“File→Import→General→Existing Projects into WorkSpace”，在弹出的对话框中，通过“Browser”按钮选择 WebCollector 文件夹，然后单击“确定”按钮后即可打开 WebCollector 项目，如图 12-8 所示。

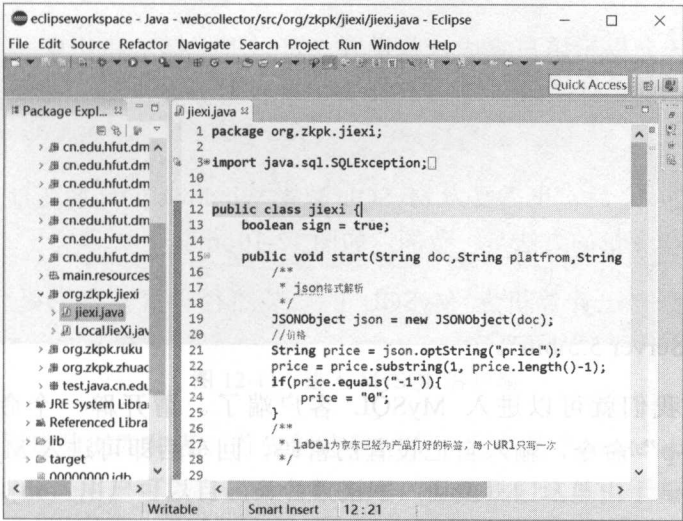


图 12-8 在 Eclipse 中导入 WebCollector 项目

12.1.4 在 Windows 下安装 MySQL

由于我们打算把爬虫获取的电商销售数据保存在本地的 MySQL 数据库中，因此这里需要安装 MySQL。

请读者从第 12 章软件资源文件夹中找到 `mysql-5.5.20-winx64.msi` 文件，双击该文件即可开始安装，安装向导的首页如图 12-9 所示。

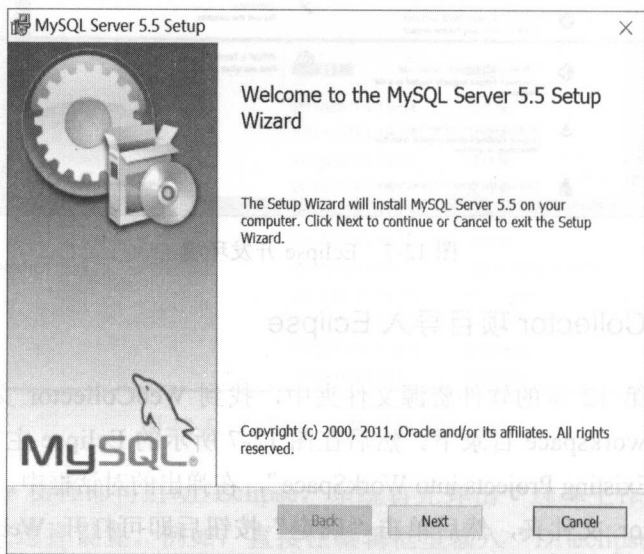


图 12-9 在 Windows 下安装 MySQL

安装中，系统会提示用户完成一些设置，可以根据提示和计算机情况进行操作。其中，在进行安全性设置时，需要提供 `root` 用户密码，请读者进行必要的设置，并记住所设置的密码。

安装好 MySQL 之后，也需要为 MySQL 配置 Windows 的环境变量，操作方法与前面介绍配置 JDK 环境变量的方法是一致的，如图 12-10 所示。

注意，要根据自己计算机上 MySQL 的安装路径设置，本书是“`C:\program files\MySQL\MySQL Server 5.5\bin`”。

完成配置，我们就可以进入 MySQL 客户端了。请开辟一个命令行窗口，执行“`MySQL -uroot -p`”命令，输入自己设置的密码，回车后即可进入 MySQL 客户端，如图 12-11 所示。

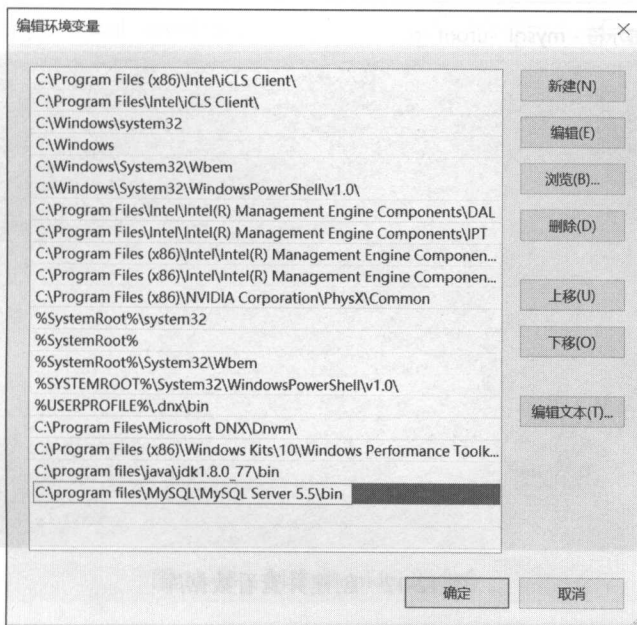


图 12-10 为 MySQL 配置 Windows 环境变量

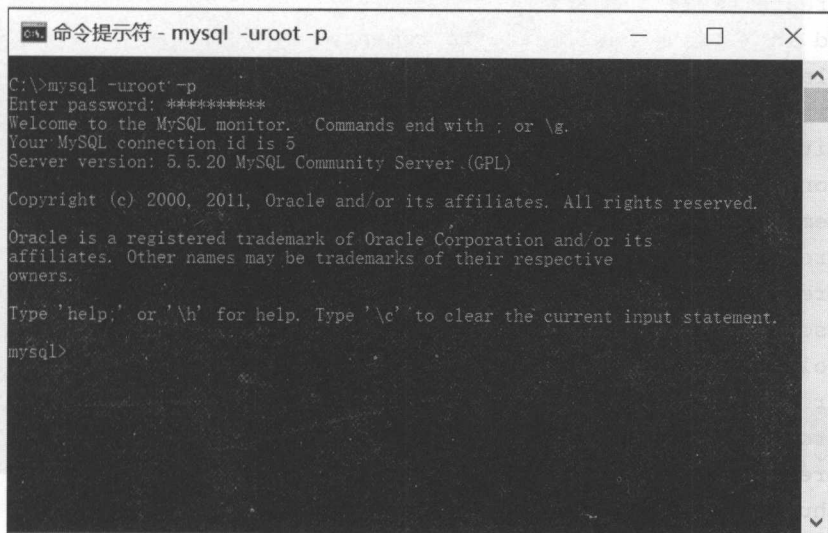


图 12-11 进入 MySQL 客户端

下面我们来创建数据库和表。首先创建数据库，命令是“create database jd_db”，其中数据库名为 jd_db。用户可以自己命名数据库，但是我们这里由于与后面的代码相关，因此名字确定为 jd_db。可以通过“show databases”命令查看数据库，如图 12-12 所示。

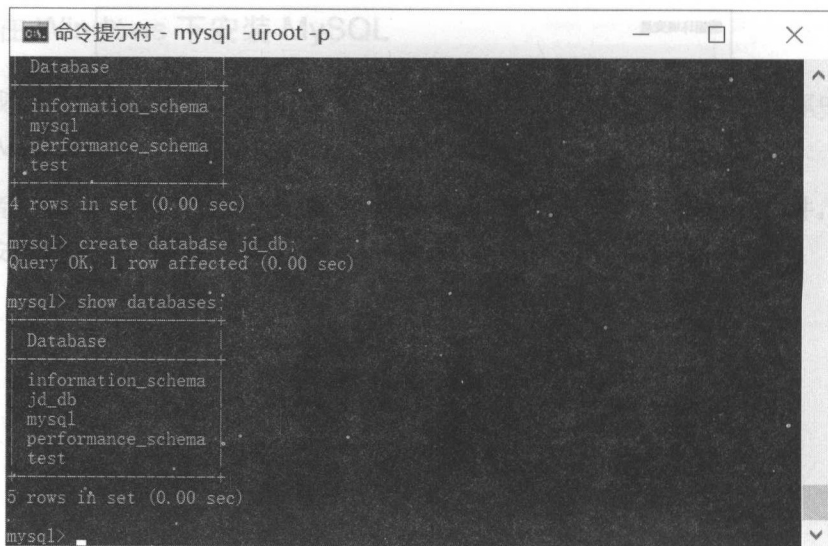


图 12-12 创建并查看数据库

接下来创建表 spider。首先执行 “use jd_db”，然后执行如下命令。

```

>create table spider(
>id int(11) not null auto_increment,
>platform varchar(255) default null,
>xinhao varchar(255) default null,
>title varchar(255) default null,
>content text default null,
>memberlevel varchar(255) default null,
>fromplatform varchar(255) default null,
>area varchar(255) default null,
>userimpression varchar(255) default null,
>color varchar(255) default null,
>price varchar(255) default null,
>productSize varchar(255) default null,
>creationTime varchar(255) default null,
>zhuoqutime varchar(255) default null,
>lable varchar(255) default null,
>primary key(id)
>)engine=MyISAM auto_increment=19712 default charset=utf8;

```

命令输入情况如图 12-13 所示。整个命令分成多行，但以 “;” 回车结束。创建成功后，可以用 “show tables” 命令查看结果，如图 12-14 所示。

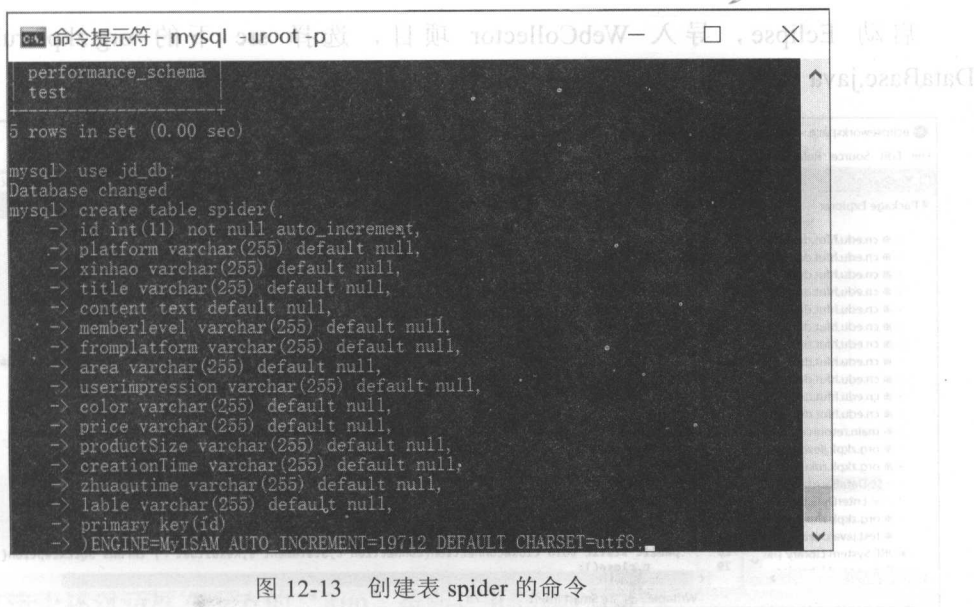


图 12-13 创建表 spider 的命令

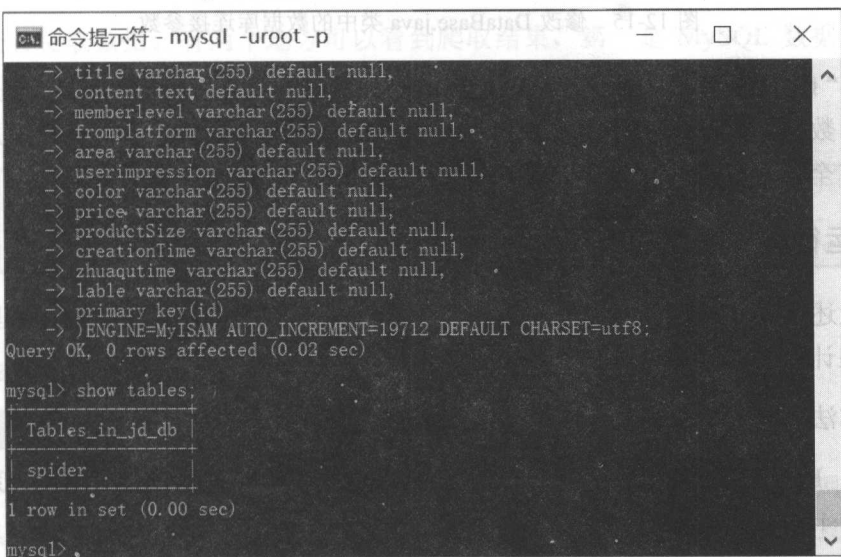


图 12-14 查看创建的 spider 表

12.1.5 连接 JDBC

WebCollector 应用程序通过 JDBC 访问 MySQL 数据库，所以，我们也需要进行相关配置。

启动 Eclipse，导入 WebCollector 项目，选择 src 下的 org.zkpk.ruku 包中的 DataBase.java 类，双击打开，然后按照图 12-15 给出的示例进行修改。

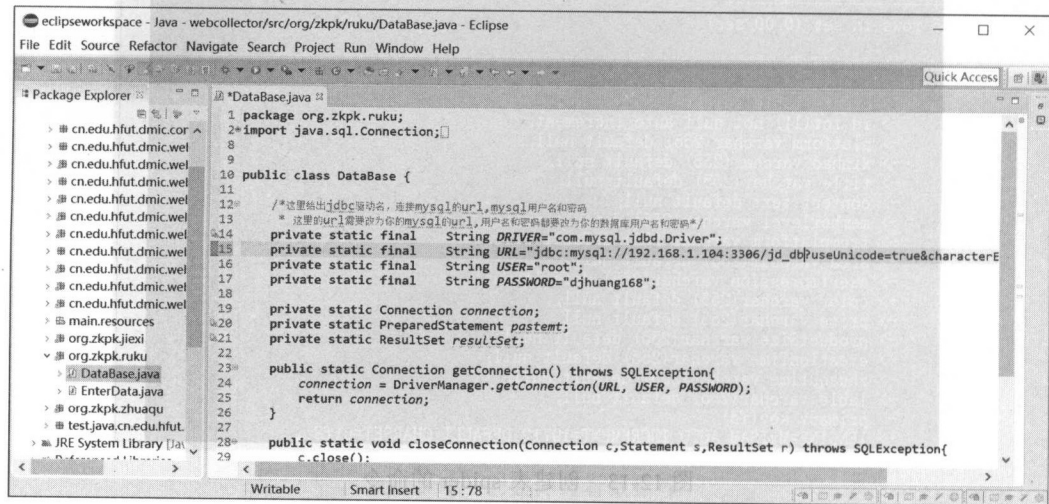


图 12-15 修改 DataBase.java 类中的数据库连接参数

其中，“192.168.1.104”是自己计算机的 IP 地址（本书是如此），“jd_db”是前面创建的 MySQL 数据库，“root”是用户名（注意，root 用户就是安装 MySQL 的用户，称为根用户，拥有全部权限），而“djhuang168”则是安装 MySQL 时设置的密码。

12.1.6 运行爬虫程序

完成上述准备工作，就可以开始运行爬虫程序抓取数据了。要成功运行这里的程序，首先要确保计算机能够访问外部 Internet。

运行方法如下。

(1) 在 Eclipse 主菜单中选择“Run”，然后选择“Run Configurations”，系统弹出对话框，如图 12-16 所示。

在上图中，需要输入运行参数，例如“2249592 喜来康 F200”。这里有三个参数，之间通过空格分开。第一个参数是一个数，2249592，表示网站上的一个品牌序列号，如“喜来康”这种按摩器品牌在京东电商平台的序列号就是 2249592；第二个参数就是品牌名称，如“喜来康”；第三个参数是型号，如 F200 喜来康按摩器的一个具体型号。这些参数是从所要爬取的网站上提取的，所以，要求开发人员事先了解页面结构和 URL 的表示方式。

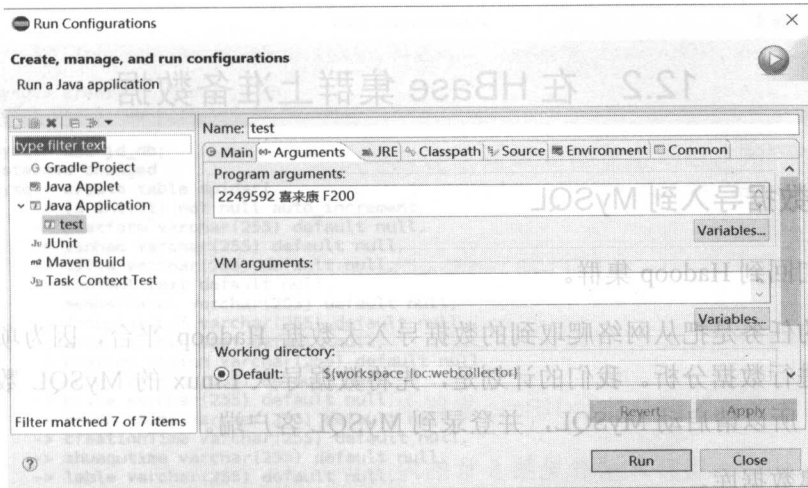


图 12-16 输入运行参数

(2) 单击上述对话框右下方的“Run”按钮，开始运行程序。

(3) 查看爬取数据。有两个地方可以看到爬取结果，第一是 MySQL 数据库；第二是 `jd_pingjia` 目录下的文本文件。例如，本书的 `jd_pingjia` 目录在 D:驱动器上，进入在目录可以看到一个以抓取日期命名的文件，如图 12-17 所示。

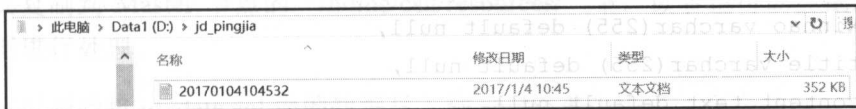


图 12-17 存储抓取数据的文本文件

双击该文件即可看到结果。

注意事项

第一，在运行上述项目的时候，被爬取的网站有可能关闭，也可能该网站部署了反爬虫系统，这种情况下自然就不能抓取网站数据。但是，有些电商平台或其他网络平台会提供一部分开放数据，通过公开的接口（如 URL）供用户访问。因此，我们可以链接这些网站来获取实际生产数据。

第二，读者可以通过 MySQL shell 查看存储在数据库（如本教程中的 `jd_db` 的 `spider` 表）中的抓取数据。但是，有时候由于某种原因（字符集设置），用户通过 `select` 这样的 SQL 语句提取表中数据并显示在 Web 页面上的时候，显示的结果是乱码。这时候需要修改 MySQL 的字符集设置，例如将字符集设置成 `utf_8`。

12.2 在 HBase 集群上准备数据

12.2.1 将数据导入到 MySQL

现在我们回到 Hadoop 集群。

接下来的任务是把从网络爬取到的数据导入大数据 Hadoop 平台，因为项目需要利用大数据平台进行数据分析。我们的计划是，先将数据导入 Linux 的 MySQL 数据库，然后导入 HBase。所以请启动 MySQL，并登录到 MySQL 客户端。

首先创建数据库。

```
mysql>create database jd_db;
mysql>use jd_db;
```

再创建 spider 表，命令如下（与在 Windows 环境下创建 MySQL 表一样）。

```
>create table spider(
>id int(11) not null auto_increment,
>platform varchar(255) default null,
>xinhao varchar(255) default null,
>title varchar(255) default null,
>content text default null,
>memberlevel varchar(255) default null,
>fromplatform varchar(255) default null,
>area varchar(255) default null,
>userimpression varchar(255) default null,
>color varchar(255) default null,
>price varchar(255) default null,
>productSize varchar(255) default null,
>creationTime varchar(255) default null,
>zhuoqutime varchar(255) default null,
>lable varchar(255) default null,
>primary key(id)
>)engine=MyISAM auto_increment=19712 default charset=utf8;
```

上述命令的输入情况如图 12-18 所示。

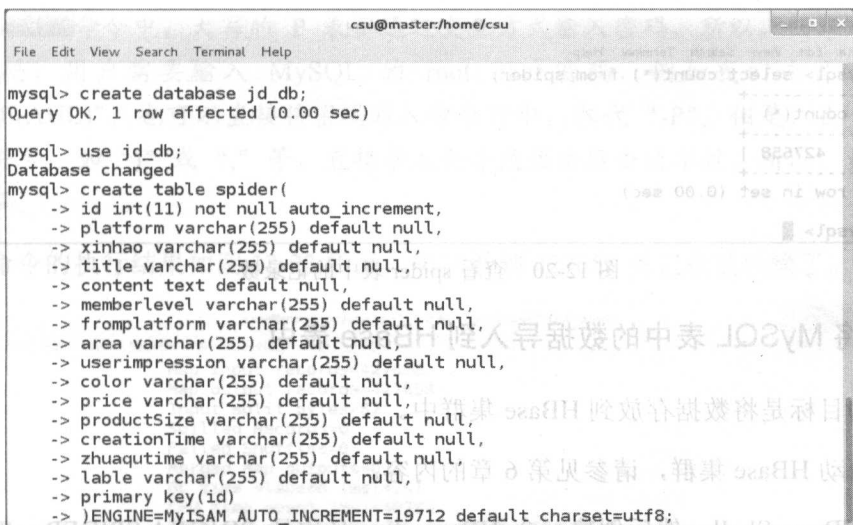


图 12-18 创建 spider 表

然后，我们向表中导入数据。请读者在本章软件资源文件夹中找到 `jd_data.sql` 文件，这是已经准备好的京东商城销售数据，有 153 MB，实际上就是通过爬虫程序抓取的真实数据。

请将其复制到 Master 节点的 `/home/csu/resources/` 下。读者也可以尝试采用自己抓取来的数据进行处理。

现在可以开始将 `jd_data.sql` 中的数据导入到 `spider` 表中去了，命令是：

```
mysql> source /home/csu/resources/jd_data.sql;
```

如图 12-19 所示。

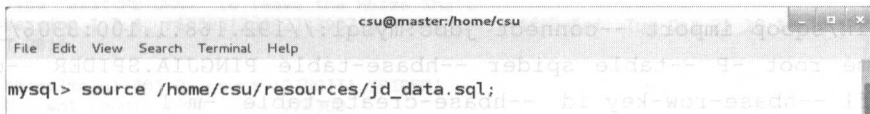


图 12-19 向 spider 表中导入数据

导入数据需要一些时间，毕竟 `jd_data.sql` 中有 427658 条记录。图 12-20 显示了查看记录数的命令与执行结果。

还可以通过 `“select * from spider limit 10”` 命令查看前 10 条记录。最好不要不加限制地查询数据，因为数据量很大，滚动显示比较耗时。

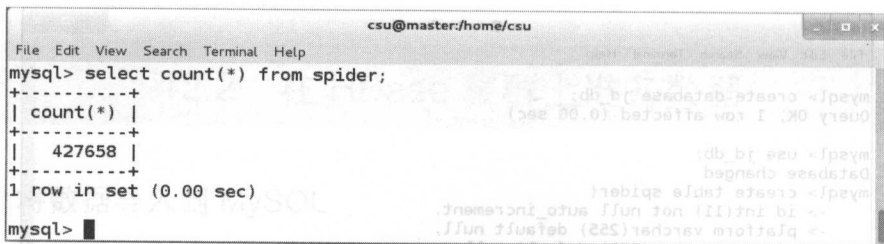


图 12-20 查看 spider 表中的记录数

12.2.2 将 MySQL 表中的数据导入到 HBase 表中

我们的目标是将数据存放到 HBase 集群中。

首先启动 HBase 集群，请参见第 6 章的内容。

进入 HBase Shell，然后创建一个 HBase 表，名称是 PINGJIA.SPIDER，整个命令如图 12-21 所示，其中“f1”是列族名。

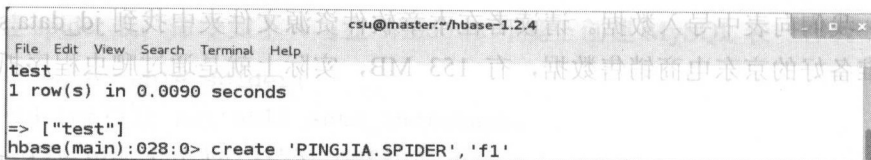


图 12-21 创建 PINGJIA.SPIDER 表

注意，HBase 的表名是区分大小写的，所以 PINGJIA.SPIDER 与 pingjia.spider 或 PINGJIA.spider 是不同的表。

接下来，我们通过 Sqoop 将 MySQL 表 spider 中的数据，导入 HBase 表 PINGJIA.SPIDER 里面。首先切换到 Sqoop 安装目录，然后执行如下命令：

```

bin/sqoop import --connect jdbc:mysql://192.168.1.100:3306/jd_db -
--username root -P --table spider --hbase-table PINGJIA.SPIDER --column-
family f1 --hbase-row-key id --hbase-create-table -m 1

```

为帮助读者正确使用上述命令，我们给出其输入的情形，如图 12-22 所示。

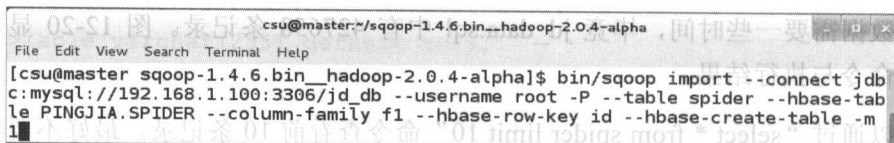


图 12-22 利用 Sqoop 从 MySQL 向 HBase 导入数据的命令

注意,上述命令中,大写的 P 表示通过交互方式输入密码,所以,回车后系统会提示输入密码,用户需要输入 MySQL 的 root 用户密码,例如我们这里的“qwerty,1289Qwdefi(AUL)”。也可以直接将密码写入命令行中,取代“-P”。但是,由于密码往往含有一些符号,如“()”或“, ”等,直接带入会导致语法检查通不过。所以,最好不要直接使用密码。

上述命令的执行结果如图 12-23 所示,可以看到 427658 条记录被传输了。

```
csu@master:~/sqoop-1.4.6/bin__hadoop-2.0.4-alpha
File Edit View Search Terminal Help
Map input records=427658
Map output records=427658
Input split bytes=87
Spilled Records=0
Failed Shuffles=0
Merged Map outputs=0
GC time elapsed (ms)=761
CPU time spent (ms)=40280
Physical memory (bytes) snapshot=195022848
Virtual memory (bytes) snapshot=912961536
Total committed heap usage (bytes)=105381888
File Input Format Counters
  Bytes Read=0
File Output Format Counters
  Bytes Written=0
16/12/15 21:05:14 INFO mapreduce.ImportJobBase: Transferred 0 bytes in 54.4666 s
econds (0 bytes/sec)
16/12/15 21:05:14 INFO mapreduce.ImportJobBase: Retrieved 427658 records.
[csu@master sqoop-1.4.6.bin__hadoop-2.0.4-alpha]$
```

图 12-23 Sqoop 命令执行完毕后的显示

为了查看 HBase 的 PINGJIA.SPIDER 表中的数据,我们可以登录 Hbase Shell,然后执行“count ‘PINGJIA.SPIDER’”命令,如图 12-24 所示。

```
csu@master:~/hbase-1.2.4
File Edit View Search Terminal Help
[csu@master hbase-1.2.4]$ bin/hbase shell
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 1.2.4, r67592f3d062743907f8c5ae00dbb1ae4f69e5af, Tue Oct 25 18:10:20 CD
T 2016
hbase(main):001:0> count 'PINGJIA.SPIDER'
Current count: 1000, row: 276300
Current count: 2000, row: 277300
Current count: 3000, row: 278300
Current count: 4000, row: 279300
Current count: 5000, row: 280300
Current count: 6000, row: 281300
Current count: 7000, row: 282300
Current count: 8000, row: 283300
Current count: 9000, row: 284300
Current count: 10000, row: 285300
Current count: 11000, row: 286300
Current count: 12000, row: 287300
```

图 12-24 在 Hbase Shell 中查看表的记录数

回车，系统滚动显示统计结果，稍候片刻（取决于数据量的大小）即可看到最后结果，如图 12-25 所示，可以看到 PINGJIA.SPIDER 表中的记录数与原来 MySQL spider 表中的记录数完全一样（427658），说明数据被成功导入到 HBase 表中。

```

csu@master:~/hbase-1.2.4
File Edit View Search Terminal Help
Current count: 413000, row: 688300
Current count: 414000, row: 689300
Current count: 415000, row: 690300
Current count: 416000, row: 691300
Current count: 417000, row: 692300
Current count: 418000, row: 693300
Current count: 419000, row: 694300
Current count: 420000, row: 695300
Current count: 421000, row: 696300
Current count: 422000, row: 697300
Current count: 423000, row: 698300
Current count: 424000, row: 699300
Current count: 425000, row: 700300
Current count: 426000, row: 701300
Current count: 427000, row: 702300
427658 row(s) in 51.4350 seconds
=> 427658
hbase(main):002:0>
  
```

图 12-25 执行 count ‘PINGJIA.SPIDER’命令的最后结果

注意事项

有这样几个错误可以导致执行上述 Sqoop 操作失败。

① 输入错误。这种错误比较容易解决，只要仔细检查输入即可。

② 使用了 localhost。有些用户在执行上述命令时不成功，给出的提示是通信连接失败（communication failed），这时候可以用 IP 地址代替 localhost，再试一下就成功了。之所以在使用 localhost 会导致连接不成功，而采用 IP 地址则可以，归根结底是没有在 Linux 的 hosts 文件中将“localhost 127.0.0.1”这项对照，修改为类似“localhost 192.168.1.100”这样的形式。

③ 没有为 Sqoop 配置 .bash_profile 文件。这也是比较常见的疏忽。请回顾第 7 章中节为 Sqoop 配置 .bash_profile 文件的方法。

上面展示的是大数据领域从 SQL 向 HBase 表导入数据的一种方法。实际上，还有两种方法，一种是 Java API 方法，另一种是 import Tsv 方法。这里就不再详细介绍，读者可以参考有关文献。

12.3 安装 Phoenix 中间件

12.3.1 Phoenix 架构

Phoenix 最早是 saleforce 的一个开源项目，后来成为 Apache 基金的顶级项目。

Phoenix 是构建在 HBase 上的一个 SQL 层，使得用户可以用标准的 JDBC APIs，而不是 HBase 客户端 APIs 来创建表、插入数据和对 HBase 数据进行查询。Phoenix 的宗旨可以用如下的口号概括：

Put the SQL back in NoSQL!

Phoenix 完全使用 Java 编写，作为 HBase 内嵌的 JDBC 驱动（Driver）供开发人员使用，Phoenix 查询引擎会将 SQL 查询转换为一个或多个 HBase 扫描（Scan），并编排执行以生成标准的 JDBC 结果集。图 12-26 给出了 Phoenix 组件在 Hadoop 生态中的位置。

Phoenix 通过以下方式使开发人员少写代码，并且性能更高（与程序员所写代码比较）：

- 将 SQL 编译成原生的 HBase scans。
- 确定 scan 关键字的最佳开始和结束。
- 让 scan 并行执行。

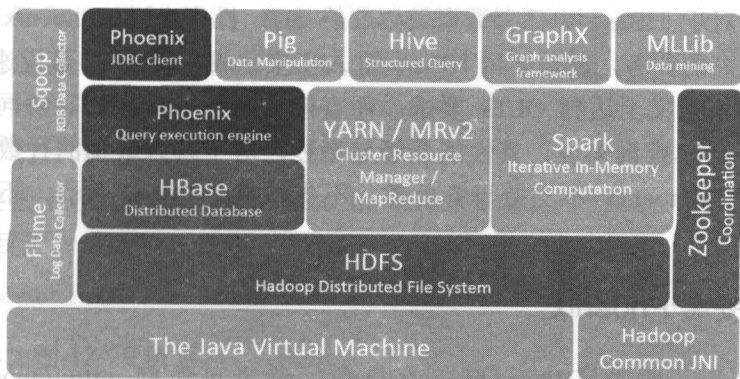


图 12-26 Phoenix 组件在 Hadoop 生态中的地位

在 Phoenix 中 SQL Query Plan 的执行，基本上是通过构建一系列的 HBase Scan 来完成的。为了尽可能减少数据的传输，Region Server 使用 Coprocessor 来执行 Aggregate 相关的工作，基本实现的思路是使用 RegionObserver 在 PostScannerOpen Hook 中将

RegionScanner 替换成支持 Aggregation 工作的定制化的 Scanner，具体的 Aggregate 操作通过 Custom 的 Scan 属性传递给 RegionScanner。与基于 MapReduce 的框架执行 Plan 的思想比较，基本上就是通过 Coprocessor 使用 RegionServer 自身来在各个节点上执行 Aggregation。另外，通过各种定制的 Filter 在 HBase 的 RegionScanner Scan 过程中，尽早地将不相关的数据过滤掉。图 12-27 是 Phoenix 运行架构。

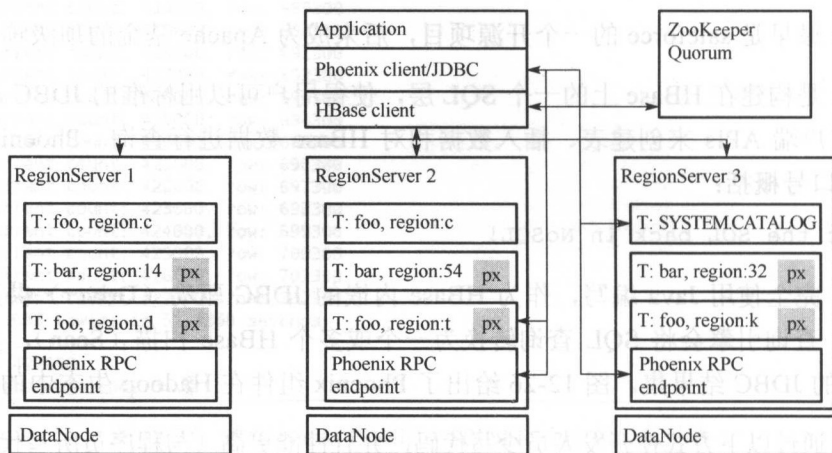


图 12-27 Phoenix 运行架构

就使用的结果来看，虽然满足了一些 CRUD (Create、Retrieve、Update 和 Delete) 的操作，然而它是在 HBase 基础上完成的，过于依赖 HBase，对其他存储介质的支持有限。总体来说，对于 HBase 中已有数据做标准的 SQL 操作来说，Phoenix 是足够了。另外，时延要求较高的业务，还是要用 HBase 的 API 来完成。Phoenix 虽说速度快、性能较高，毕竟不能严格意义上达到 OLTP 要求。而直接使用 HBase API、协同处理器与自定义过滤器。对于简单查询来说，其性能量级是毫秒，对于百万级别的行数来说，其性能量级是秒。总之，Phoenix 不支持事务处理，不支持复杂的条件查询，不支持表之间的关联操作。

12.3.2 解压安装 Phoenix

安装 Phoenix，要求 HBase 已正常启动。读者可访问 <http://phoenix.apache.org/download.html> 下载 Phoenix: apache-phoenix-4.9.0-HBase-1.2-bin.tar，也可以在本书第 10 章软件资源文件夹中找到该文件。

请将该文件复制到 Master 节点的“/home/csu/”目录下，执行如下解压缩命令：

```
tar -zxvf apache-phoenix-4.9.0-HBase-1.2-bin.tar.gz
```


上述命令输入情形如图 12-28 所示。

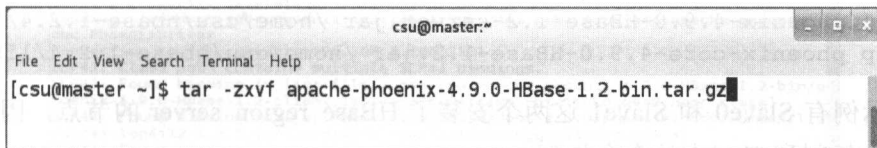


图 12-28 执行 count ‘PINGJIA.SPIDER’命令的最后结果

解压缩完毕，系统自动生成 Phoenix 主安装目录。读者可以进入该目录查看文件内容。

12.3.3 Phoenix 环境配置

1. 修改 Linux 环境变量

执行“gedit /home/csu/.bash_profile”命令，将如下代码放进.bash_profile 文件。

```
export PHOENIX_HOME=/home/csu/apache-phoenix-4.9.0-HBase-1.2-bin
export PATH=$PHOENIX_HOME/bin:$PATH
```

输入完毕，保存并退出 gedit 编辑器。

执行“source /home/csu/.bash_profile”命令，使上述配置生效。

2. 复制依赖库

HBase 在与 Phoenix 结合中，需要利用对应的依赖包，这些文件如下。

```
phoenix-4.9.0-HBase-1.2-client.jar
phoenix-4.9.0-HBase-1.2-server.jar
phoenix-core-4.9.0-HBase-1.2.jar
```

我们需要把上述文件复制到 Master 和 Slave 节点上的 HBase lib 目录中。首先进入 Phoenix 安装目录，然后执行复制到 Master 节点的操作命令。

```
cp phoenix-4.9.0-HBase-1.2-client.jar /home/csu/hbase-1.2.4/lib/
```

输入情形如图 12-29 所示。

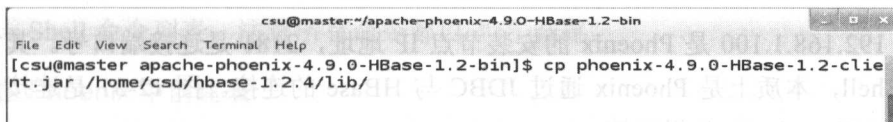


图 12-29 复制 phoenix-4.9.0-HBase-1.2-client.jar 文件到 Master 节点的 HBase lib 目录中

其他两个文件的复制命令类似：

```
cp phoenix-4.9.0-HBase-1.2-server.jar /home/csu/hbase-1.2.4/lib/
cp phoenix-core-4.9.0-HBase-1.2.jar /home/csu/hbase-1.2.4/lib/
```

本书示例有 Slave0 和 Slave1 这两个安装了 HBase region server 的节点，因此要进行两批复制。复制到 Slave0 的命令如下：

```
scp -r phoenix-4.9.0-HBase-1.2-client.jar csu@slave0:/home/csu/
hbase-1.2.4/lib/
scp -r phoenix-4.9.0-HBase-1.2-server.jar csu@slave0:/home/csu/
hbase-1.2.4/lib/
scp -r phoenix-core-4.9.0-HBase-1.2.jar csu@slave0: /home/csu/
hbase-1.2.4/lib/
```

其中第一条命令的输入情形如图 12-30 所示。

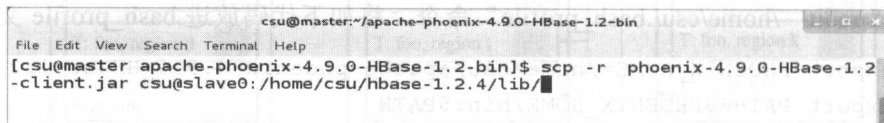


图 12-30 复制 phoenix-4.9.0-HBase-1.2-client.jar 文件到 Slave0 节点的 HBase lib 目录中

其他两条命令的输入类似。

3. 重启 HBase 集群

进入 Master 节点的 HBase 安装目录，执行如下命令。

```
bin/stop-hbase.sh
bin/start-hbase.sh
```

12.3.4 使用 Phoenix

1. Phoenix Shell 基本命令

要进入 Phoenix Shell，必须首先启动 Hadoop 和 HBase。在此条件下，执行如下命令即可进入 Phoenix Shell。

```
bin/sqlline.py 192.168.1.100:2181
```

其中 192.168.1.100 是 Phoenix 的安装节点 IP 地址，2181 是连接端口号。其实，启动 Phoenix Shell，本质上是 Phoenix 通过 JDBC 与 HBase 的连接。图 12-31 是连接成功后的显示信息与 Phoenix Shell 提示符。

```

csu@master:~/apache-phoenix-4.9.0-HBase-1.2-bin
File Edit View Search Terminal Help
Setting property: [isolation, TRANSACTION_READ_COMMITTED]
issuing: !connect jdbc:phoenix:192.168.1.100:2181 none none org.apache.phoenix.j
dbc.PhoenixDriver
Connecting to jdbc:phoenix:192.168.1.100:2181
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/csu/apache-phoenix-4.9.0-HBase-1.2-bin/p
hoenix-4.9.0-HBase-1.2-client.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/csu/hadoop-2.6.0/share/hadoop/common/lib
/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/csu/hbase-1.2.4/lib/phoenix-4.9.0-HBase-
1.2-client.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
16/12/19 10:12:23 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
Connected to: Phoenix (version 4.9)
Driver: PhoenixEmbeddedDriver (version 4.9)
Autocommit status: true
Transaction isolation: TRANSACTION_READ_COMMITTED
Building list of tables and columns for tab-completion (set fastconnect to true
to skip)...
87/87 (100%) Done
Done
sqlline version 1.2.0
0: jdbc:phoenix:192.168.1.100:2181>

```

图 12-31 启动 Phoenix Shell

注意事项

启动 Phoenix Shell 时可能出现异常，主要原因有：

第一，ZooKeeper 没有启动。无论是 HBase 自带的 ZooKeeper 还是独立安装的 ZooKeeper，只要能够确保正常启动即可投入应用。我们这里是通过启动 HBase 来启动 ZooKeeper 的。

第二，执行命令“./sqlline.py”时，后面使用了主机名或 localhost。由于配置上的差异，有些用户采用主机名会报错，这是因为其 Hadoop 配置没有包含主机名与 IP 地址的映射。所以，如果用主机名或 localhost 连接不成功，就改用 IP 地址试一下。

第三，Phoenix 和 HBase 兼容问题。读者注意到，apache-phoenix-4.9.0-HBase-1.2-bin.tar.gz 文件名中包含了 Phoenix 和 HBase 版本号，这就表明了它们之间的版本兼容性配置。如果安装的 HBase 是 0.98，而使用 phoenix-4.9.0，就有可能出现兼容性问题。

第四，Phoenix 的依赖包没有被拷贝到 HBase 的 lib 目录下。这是操作问题。

进入 Phoenix Shell 后，就可以输入命令使用 Phoenix 了。用户不妨输入 help 命令查看 Phoenix Shell 命令列表，注意所有命令都以“;”结尾。

2. 在 Phoenix 中创建表

下面我们在 Phoenix 中创建“PINGJIA.SPIDER”表，该表与 HBase 中的 PINGJIA.SPIDER 表对应。

请在 Phoenix Shell 提示符下执行如下命令。

```
create table PINGJIA.SPIDER(
  id varchar primary key,
  "f1"."platform" varchar,
  "f1"."xinhao" varchar,
  "f1"."title" varchar,
  "f1"."content" varchar,
  "f1"."memberlevel" varchar,
  "f1"."formplatform" varchar,
  "f1"."area" varchar,
  "f1"."userimpression" varchar,
  "f1"."color" varchar,
  "f1"."price" varchar,
  "f1"."productSize" varchar,
  "f1"."creationTime" varchar,
  "f1"."zhuaqutime" varchar,
  "f1"."lable" varchar);
```

输入完毕回车后，系统开始执行上述命令。等待片刻后，即可看到执行成功后的显示信息，如图 12-32 所示。



```
csu@master:/apache-phoenix-4.9.0-HBase-1.2-bin
File Edit View Search Terminal Help
0: jdbc:phoenix:192.168.1.100:2181> create table PINGJIA.SPIDER(
. . . . . > id varchar primary key,
. . . . . > "f1"."platform" varchar,
. . . . . > "f1"."xinhao" varchar,
. . . . . > "f1"."title" varchar,
. . . . . > "f1"."content" varchar,
. . . . . > "f1"."memberlevel" varchar,
. . . . . > "f1"."fromplatform" varchar,
. . . . . > "f1"."area" varchar,
. . . . . > "f1"."userimpression" varchar,
. . . . . > "f1"."color" varchar,
. . . . . > "f1"."price" varchar,
. . . . . > "f1"."productSize" varchar,
. . . . . > "f1"."creationTime" varchar,
. . . . . > "f1"."zhuaqutime" varchar,
. . . . . > "f1"."lable" varchar);
427,658 rows affected (29.991 seconds)
0: jdbc:phoenix:192.168.1.100:2181>
```

图 12-32 在 Phoenix 中创建 PINGJIA.SPIDER 表

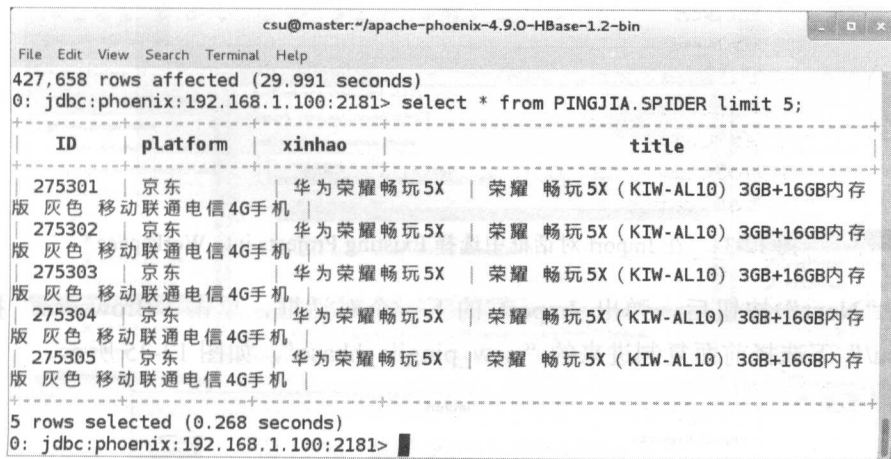
可以看出，有 427658 行记录被关联（Affected）了。可以看出，Phoenix 本质上是一种 HBase 的查询工具。在 Phoenix 中创建表，实际上是通过 JDBC 建立与 HBase 表一个的连接，而数据仍然是存储在 HBase 中的，但现在可以通过 Phoenix 操作 HBase 表了。因此，这就要求 Phoenix 中创建的表与用户所希望操作的 HBase 表存在对应关系，这种对应必须完全一致，例如，表名必须相同，并且大小写也要一致，字段名完全一致。

HBase 的查询工具具有很多, 除 Phoenix 外, 还有 Hive、Tez、Impala 和 Spark SQL 等。我们这里之所以介绍 Phoenix, 主要目的还是向大家展示 Phoenix 的应用。不过, Phoenix 在众多的竞争者中有它的优势, 因此可以在生产应用中加以考虑。

下面我们通过 Phoenix 来查看 HBase 中的 PINGJIA.SPIDER 表的内容。Phoenix 的查询命令与标准 SQL 一致, 例如查看 PINGJIA.SPIDER 表中的前 5 条记录。

```
select * from PINGJIA.SPIDER limit 5;
```

图 12-33 给出了上述命令的执行结果。



```
csu@master:~/apache-phoenix-4.9.0-HBase-1.2-bin
File Edit View Search Terminal Help
427,658 rows affected (29.991 seconds)
0: jdbc:phoenix:192.168.1.100:2181> select * from PINGJIA.SPIDER limit 5;
```

ID	platform	xinhao	title
275301	京东	华为荣耀畅玩5X	荣耀 畅玩5X (KIW-AL10) 3GB+16GB内存
275302	京东	华为荣耀畅玩5X	荣耀 畅玩5X (KIW-AL10) 3GB+16GB内存
275303	京东	华为荣耀畅玩5X	荣耀 畅玩5X (KIW-AL10) 3GB+16GB内存
275304	京东	华为荣耀畅玩5X	荣耀 畅玩5X (KIW-AL10) 3GB+16GB内存
275305	京东	华为荣耀畅玩5X	荣耀 畅玩5X (KIW-AL10) 3GB+16GB内存

```
5 rows selected (0.268 seconds)
0: jdbc:phoenix:192.168.1.100:2181>
```

图 12-33 在 Phoenix 中查询 PINGJIA.SPIDER 表

要查看具体字段内容, 可以使用如下命令。

```
select "content" from PINGJIA.SPIDER limit 100;
```

Phoenix Shell 还有很多命令, 读者如果感兴趣可以参考有关文献或网站。

12.4 基于 Web 的前端开发

12.4.1 将 Web 前端项目导入 Eclipse

我们在第 5 章已经为 Linux 安装了 Eclipse, 现在再次启动该工具。

在 Eclipse 主菜单中, 选择 “File→Import→General→Existing Projects into Workspace→Next”, 如图 12-34 所示。

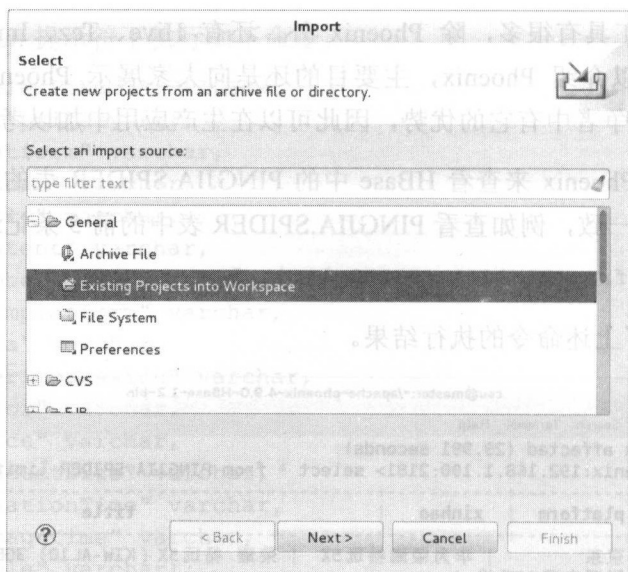


图 12-34 在 Import 对话框中选择 Existing Projects into Workspace

单击“Next”按钮后，弹出 Import 的下一个对话框，单击“Browse...”按钮，在“/home/csu/”下选择前面复制进来的“new_pingjia_hbase”，如图 12-35 所示。

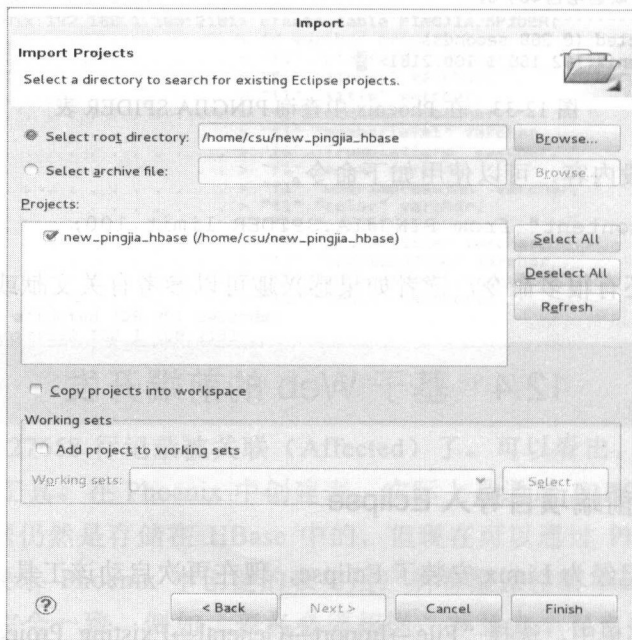


图 12-35 通过 Browse 选择项目所在目录

单击图 12-35 中的“Finish”按钮即可导入项目。

上面导入的工程是某公司的一个示范性项目，源代码可以供读者研究和学习（但是不能用于商业目的）。现在，我们在这里仅对源码进行一个简单的修改。

在 Eclipse 的 Project Explorer 中展开“src”，打开 phoenix_Hbase.java 文件，修改其中的 private static final string URL，将 IP 地址替换成自己计算机 Master 节点的 IP 地址（例如 192.168.1.100，请根据计算机具体情况修改），如图 12-36 所示。

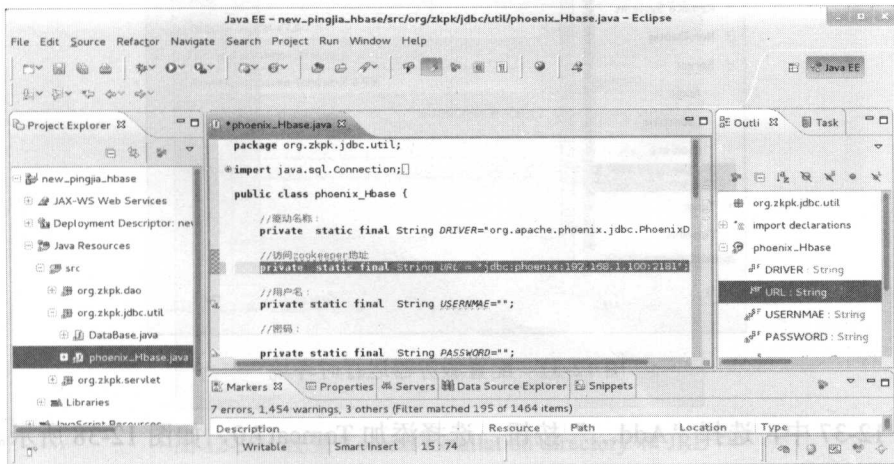


图 12-36 编辑 phoenix_Hbase.java 文件

至此，代码修改完毕。

12.4.2 安装 Tomcat

接下来我们需要安装 Tomcat。请读者在本章软件资源文件夹中找到 apache-tomcat-7.0.59.zip 文件，将其复制到 Master 节点的“/home/csu/”目录下。然后执行如下的解压缩命令。

```
unzip apache-tomcat-7.0.59.zip
```

执行成功后，系统在“/home/csu/”下，自动创建了 apache-tomcat-7.0.59 目录，此即为 Tomcat 的主安装目录。

12.4.3 在 Eclipse 中配置 Tomcat

切换到 Eclipse 界面，在主菜单中选择“Window→Preferences→Server→Runtime Environments”，如图 12-37 所示。

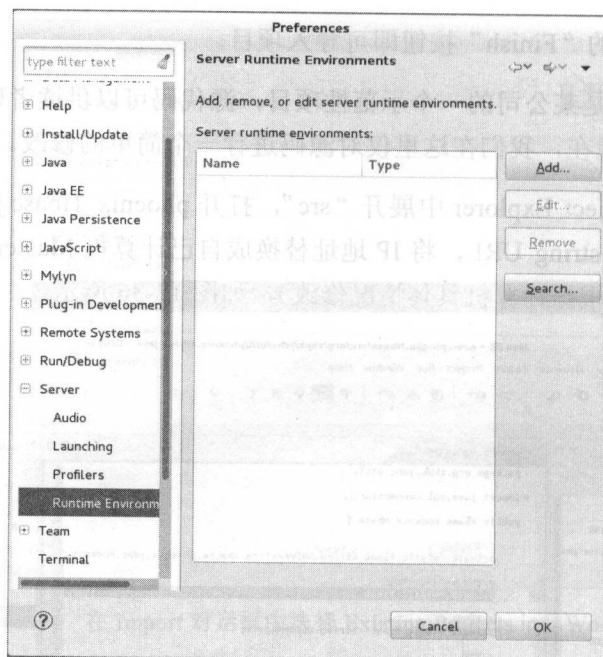


图 12-37 配置服务器运行时环境

在图 12-37 中, 选择 “Add...” 按钮, 选择添加 Tomcat7.0, 如图 12-38 所示。

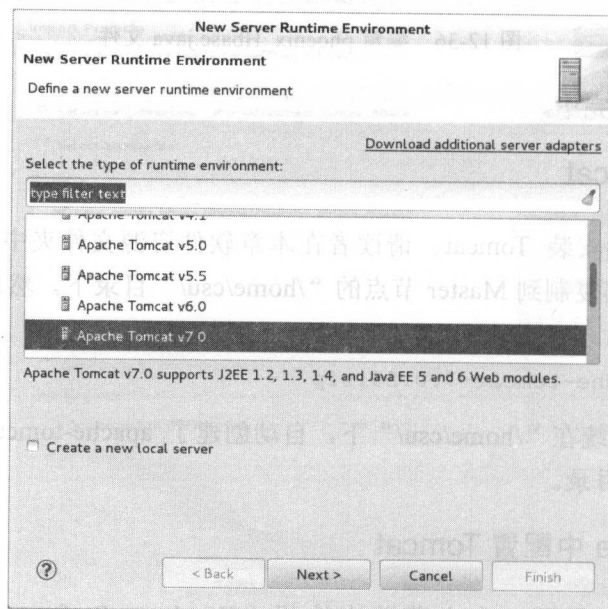


图 12-38 选择添加 Tomcat7.0

单击图 12-38 中的“Next>”按钮，弹出如图 12-39 所示的“Tomcat Server”对话框。在该对话框中，我们需要指定 Tomcat 的安装目录，并选择 JRE（Java Runtime Environments）。前者通过单击图 12-39 中的“Browse...”按钮进行选择，后者只要单击 JRE 下面的下拉式列表，选择其中的“jdk1.7.0_71”即可。

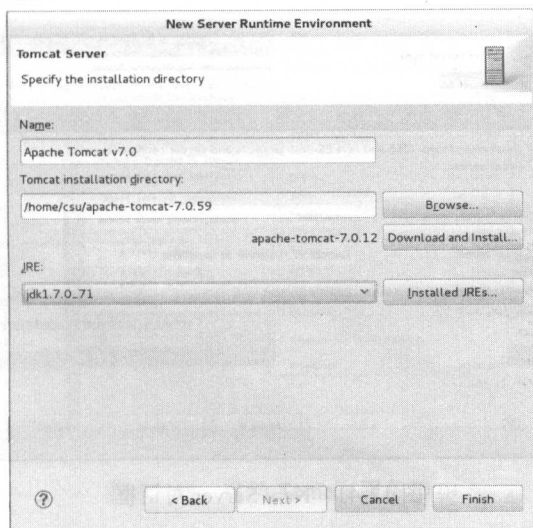


图 12-39 配置 Tomcat installation directory 和 JRE

完成选择后，单击“Finish”按钮，再单击返回页面的“OK”按钮，即可完成了配置。然后回到 Eclipse 主界面，打开“Server”页面，如图 12-40 的中下部所示。

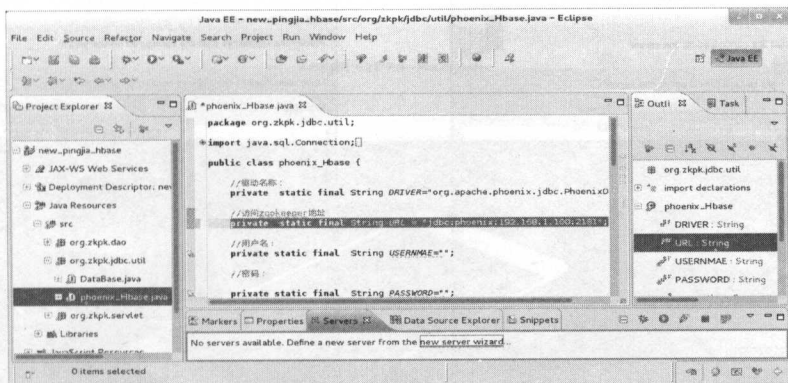


图 12-40 打开主界面中的 Server 页面

如果 Server 页面事先没有出现在主界面中，也可以通过主菜单的“Window→Show View→Others→Server”选择将其打开。

单击图 12-40 中的“new server wizard”链接，开始添加 Tomcat。系统首先弹出“New Server”对话框，如图 12-41 所示。

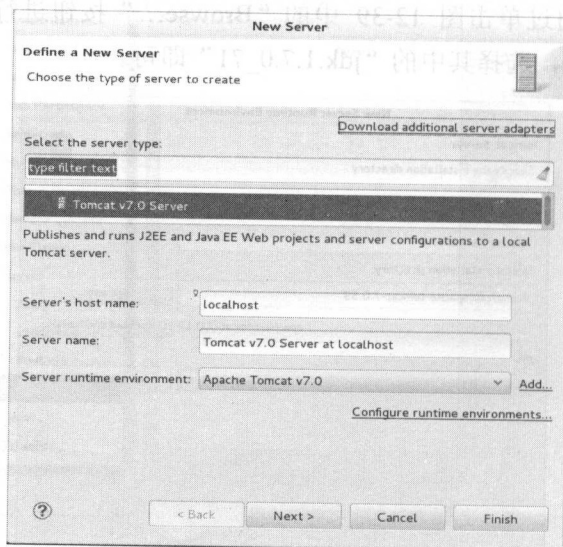
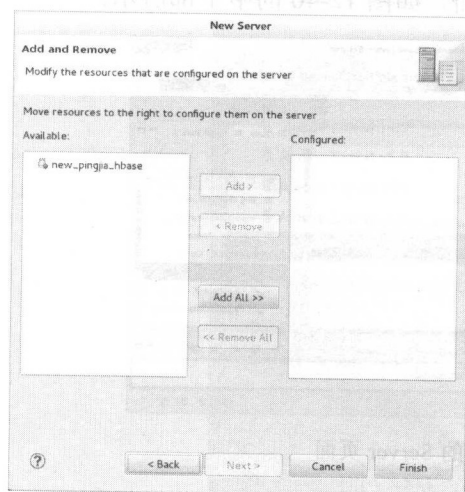
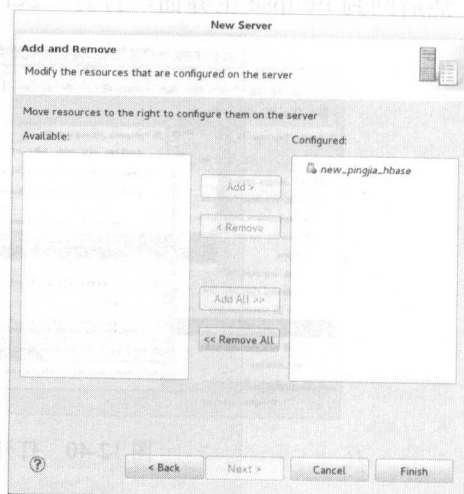


图 12-41 New Server 对话框

直接单击上述对话框中的“Next”按钮。在随后弹出的“Add and Remove”对话框中，首先选择左边的“new_pingjia_hbase”，再单击中间的“Add>”，即可将项目配置到服务器中去，如图 12-42 所示。



(a)



(b)

图 12-42 为 new_pingjia_hbase 项目配置 Tomcat 服务器

单击图 12-42 (b) 中的“Finish”按钮，即完成服务器的添加。

最后启动 Tomcat。方法是，在 Eclipse 主界面的“Server 页面中”，选择“Tomcat 7.0 Server at localhost[Stopped, Synchronized]”，单击鼠标右键，在弹出的列表中选择“Start”，即可开始运行项目，如图 12-43 所示。

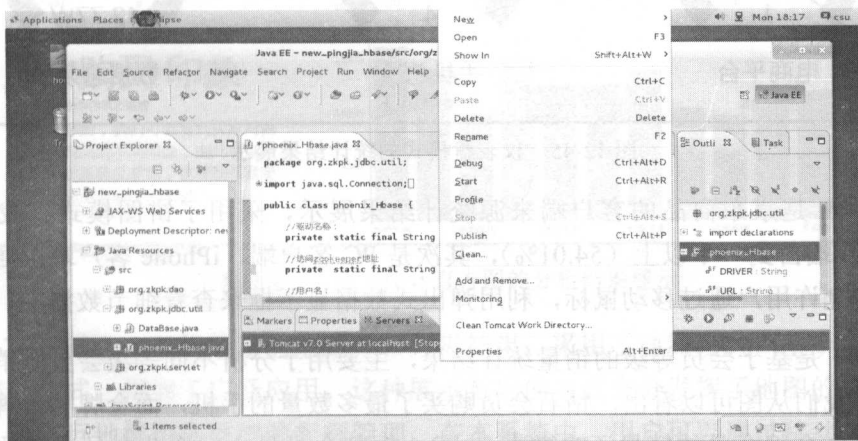


图 12-43 启动 Tomcat 服务器

12.4.4 在 Web 浏览器中查看执行结果

完成上述操作后，我们就可以通过 Firefox 浏览器查看程序运行结果了。请在浏览器地址栏输入 http://master:8080/new_pingjia_hbase/index.jsp，回车后将看到如图 12-44 所示的页面。



图 12-44 new_pingjia_hbase 项目首页

图 12-45 是电商平台数、手机总量、数据总量的统计结果的展示，采用了仪表盘模式。



图 12-45 仪表盘模式的统计结果展示

图 12-46 是京东商品的客户端来源统计结果展示，采用了饼图模式。我们看到，Android 客户端占到一半以上（54.01%），其次是 PC 客户端，iPhone 客户端位居第三。注意，该项目允许用户通过移动鼠标，利用弹出式数据显示框来查看细节数据。

图 12-47 是基于会员等级的销量统计结果，主要用于分析不同等级会员对销售的倾向性，例如，我们从图可以看出，钻石会员购买了最多数量的手机，而金牌与银牌会员的销量差别不大（金牌会员的销量还略微少一点），铜牌会员最少，企业会员与其他注册会员的销量很小。

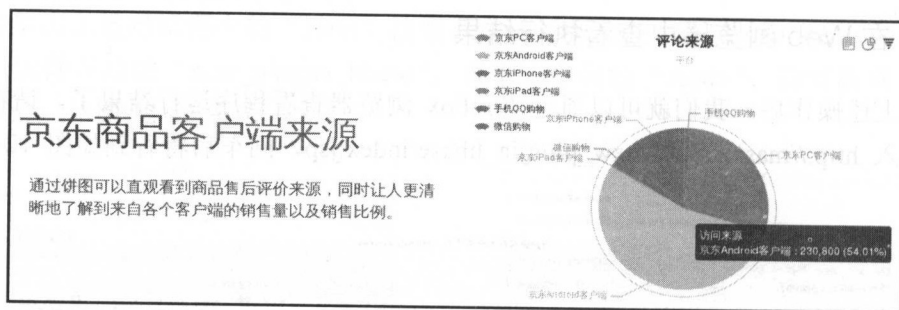


图 12-46 饼图模式的统计结果展示

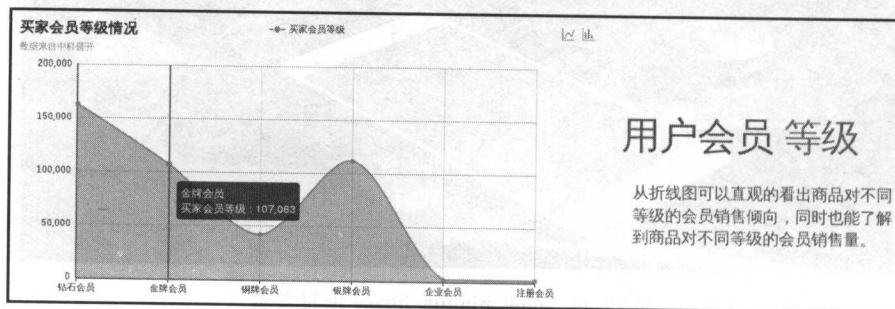


图 12-47 曲线形式的统计结果展示

图 12-48 给出了用户印象的分析结果, 采用了统计直方图模式。从显示结果可以直观地看出, 用户对商品的印象分布, 其中性价比最为用户所关心, 其次是系统流畅度和外观等。显然, 这些分析结果有助于手机制造商改进手机设计, 也有助于销售平台进行营销管理。

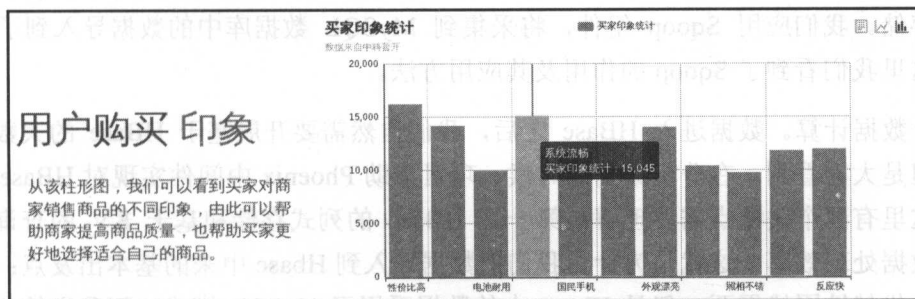


图 12-48 基于统计直方图的分析结果展示

图 12-49 是京东手机销量的地域分布分析结果, 采用了地图模式。基于地图的大数据分析可视化模式, 得到了广泛应用。这种展示数据的方式充分发挥了地图的优势, 能够支持决策人员进行地域销售渠道的配额管理。在本系统中, 用户可以通过移动鼠标到不同的省份区域, 通过弹出的小窗口查看对应省份的销售量。

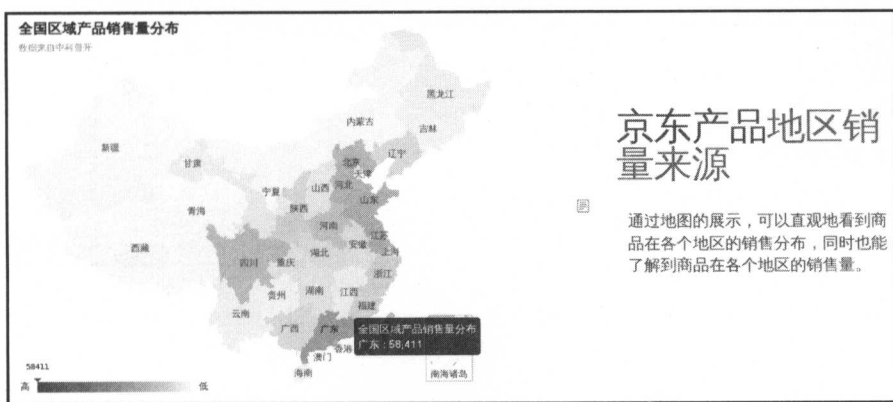


图 12-49 基于地理信息的分析结果展示

12.5 本章小结

本章介绍了大数据分析应用系统的完整开发过程, 涵盖了数据采集、数据转换、数据分析和结果展示的整个项目流程。

(1) 数据采集。在数据采集方面，我们给出了基于 WebCollector 的爬虫系统设计方法，展示了将 Web 网页数据抓取、分析并存储到 MySQL 数据库的基本实现过程。

(2) 数据存储。有了数据之后，我们还必须将采集到的数据导入到 Hadoop 大数据分析平台存储。我们应用 Sqoop 组件，将采集到 MySQL 数据库中的数据导入到了 HBase 集群。这里我们看到了 Sqoop 的作用及其应用方法。

(3) 数据计算。数据进入 HBase 之后，我们自然需要开展基于 HBase 的大数据分析计算，但是大家看到，在分析数据的时候，项目借助 Phoenix 中间件实现对 HBase 数据的读取。这里有两个关键点需要理解：第一是 HBase 的列式存储和基于 KV 的查询，大大提高了数据处理效率，这就是为什么我们把数据导入到 Hbase 中来的基本出发点；第二，HBase 的优越性固然需要，但是 Hbase 中的数据采用了 NoSQL 模式，而我们的分析系统则是基于传统结构化数据库模式的，因此需要通过一个“Put the SQL back in NoSQL!”的中间件，将 HBase 数据转换成标准的 SQL 分析数据，这就是 Phoenix 的作用。

(4) 数据分析和展示。实际上，在我们这个项目中，数据分析和展示是通过后台的 Java Servlet 程序实现的。读者可以分析代码具体了解实现技术。



Hadoop

大数据实战权威指南

大数据贵在落实!

把原理、架构、运行流程分析与实际应用融合起来介绍,因为原理最终要付诸应用;高度重视实践能力的培养,详细描述了系统安装、配置和应用过程,所有实验都是基于实际完成的操作介绍的;突出了数据处理本身,深入介绍如何运用技术进行实际的数据分析,采用的数据样本来自实际,展示的项目具有实用参考价值。

第一篇主要介绍大数据的背景、发展及关键技术;第二篇为Hadoop大数据平台搭建与基本应用,内容涉及Linux、HDFS、MapReduce、YARN、Hive、HBase、Sqoop、Kafk、Spark等;第三篇为大数据处理与项目开发,包括交互式数据处理、协同过滤推荐系统、销售数据分析系统,并就京东的部分销售数据应用大数据进行处理分析。



电子信息出版分社微博
<http://weibo.com/etpublish>



责任编辑:田宏峰
封面设计:汇众智捷



官方微信平台

ISBN 978-7-121-31821-4



9 787121 318214 >

定价: 68.00元